COMMODORE MAGAZINE

VOL. 2
ISSUE 7

★ New VIC Games
★ SuperPET Printers

Registered by Australia Post
Publication No. NBP4288

REGISTERED FOR POSTING AS A PUBLICATION: CATEGORY B

The objective of this magazine is to disseminate information to all users of Commodore computer products. This magazine contains a variety of information collected from other Commodore publications, and generated locally. Contributions from all Commodore User Groups, and individual users are encouraged.

Advertising space is restricted and allocated on a first-come, first-served basis. Advertising rates can be supplied on request.

All copy and advertising should be addressed to:

The Editor,
COMMODORE MAGAZINE,
P.O. BOX 336, ARTARMON
N.S.W. 2064
AUSTRALIA

ISSUE No.	COPY/ADV DEADLINE	PUBLICATION DATE
1	February 18th	March 6th
2	April 1st	April 17th
3	May 13th	May 29th
4	June 17th	July 3rd
5	July 29th	August 14th
6	September 9th	September 25th
7	October 14th	October 30th
8	November 25th	December 4th

Production & Typesetting: Mervyn Beamish Graphics Pty. Ltd. 82 Alexander Street, Crows Nest 2065 Phone 439 1827

Printer: Liberty Print 108 Chandos Street, Crows Nest 2067 Phone 43 4398

SUBSCRIPTIONS

	Annual Subscription	Single Copy
Postage paid within Australia	$A30.00	$A5.00
Overseas Postage Paid	$A38.00	$A6.00

Subscriptions to COMMODORE MAGAZINE can be obtained from your authorised Commodore Dealer, or from:

COMMODORE MAGAZINE,
P.O. BOX 336, ARTARMON,
N.S.W. 2064,
AUSTRALIA.

Vol 1 1981
Vol 2 1982

Typeset and assembled off Commodore Wordcraft disks

PLEASE NOTE: To provide a good information service to Commodore users, we will regularly mention equipment, software and services offered by companies and individuals not directly related to Commodore. In doing so, we are not making recommendations, and cannot be responsible for the validity and accuracy of any statements made.

EDITOR'S DESK . . .

Commodore International Limited, the parent company of Commodore Business Machines Pty. Ltd. in Australia, has released the financial results for fiscal year 1982 ended on June 30th. It is pleasing to see the record sales and the continuing growth rate, despite the so called economic ressions. The same trends are also applicable in Australia and New Zealand.

Commodore is committed to a continued growth pattern and the large percentage of sales that is being reinvested into research and development will help to realise this goal.

To quote from The Commodore Philosophy, 'Commodore is driven by technology. We don't introduce only the products the customer wants... we introduce products the customer didn't even know were available.

'We're always looking at the future because we're helping to create that future... but the work is always done in the present.'

table of contents

next issue:

News on The Commodore 64
SuperPET Update
Subscription Renewals

list of advertisers

COMMODORE NEWS

DATA 82 WIN A VIC-20 COMPETITION

At a recent computer show in Sydney, a VIC-20 computer was offered as a prize for the first entry drawn from the barrel with correct answers to some simple questions.

The winner was Graham Martin from Blackburn in Victoria. Graham works with AUSTPAK and was hoping to use the VIC-20 as a terminal, but his family may have other ideas to get into general computing.

Graham (left) is shown being presented with the VIC-20 by Neil Brandie, General Manager of Panatronics Pty. Ltd., a Commodore Dealer at Mont Albert.

COMMODORE ANNOUNCES INCREASED SALES AND PROFITS

The fiscal year ended June 30, 1982, represented the fifth consecutive year of record sales, net income and earnings per share for Commodore International Limited.

During fiscal 1982, sales of microcomputer systems accounted for 75% of overall Commodore sales compared to 71% in fiscal 1981 and 66% in fiscal 1980.

In Europe and other non-U.S. countries, Commodore continued to maintain its leading market position in the sales of microcomputer systems.

Commodore's Semiconductor Components Division continues to serve as the nucleus of the Company's vertically integrated structure. Many of the microprocessors and ROM's produced by Commodore's semiconductor facilities were utlized by Commodore in its own microcomputer systems and software cartridges. The remainder of the production from this division was sold to outside customers and accounted for 20% of Commodore's overall sales.

The Consumer Products and Office Equipment Divisions combined, contributed approximately 5% to Commodore's fiscal 1982 sales.

Commodore's Office Equipment Division continued to enjoy a leading market position in Canada, while at the same time continuing to supply Commodore with thousands of metal housings used in the Company's microcomputer systems.

The Consumer Products Division sells watches and electronic thermostats which provide Commodore with an outlet for some of its liquid crystal display (LCD) output.

The fiscal 1982 research and development outlay was $17.9 million, or 5.9% of sales.

The primary thrust of Commodore's reasearch and development continues to be in three major areas; Hardware Systems Design, Software Systems Design, and Semiconductor Chip Design for advanced very large scale integrated circuits.

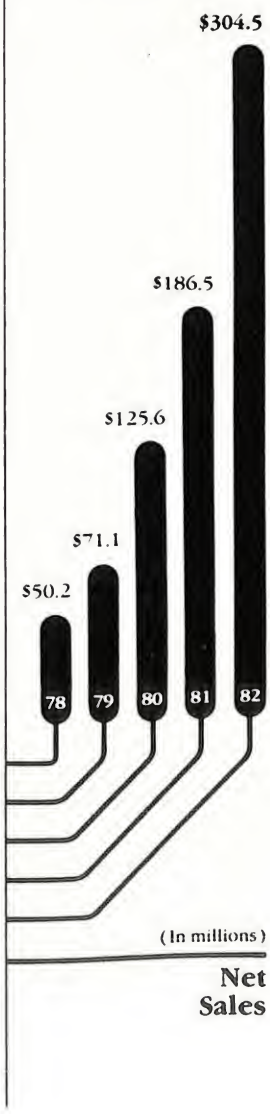
For 1983 a series of new proprietary systems are planned, including a family of advanced microprocessors and peripheral integrated circuits for high speed, low power battery operated microcomputer systems, and improved video graphics relative to anything offered previously.

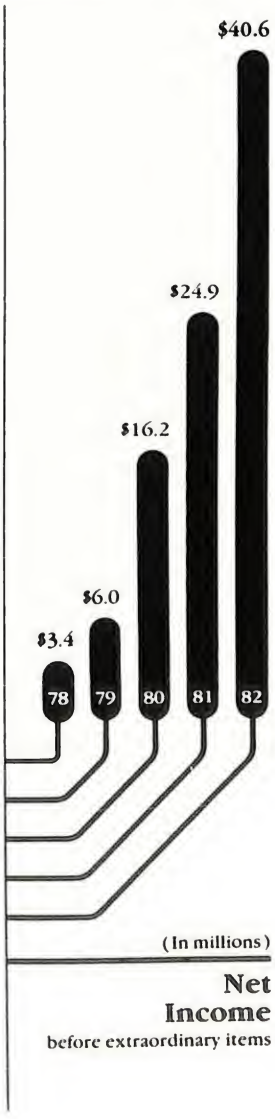
Financial Highlights

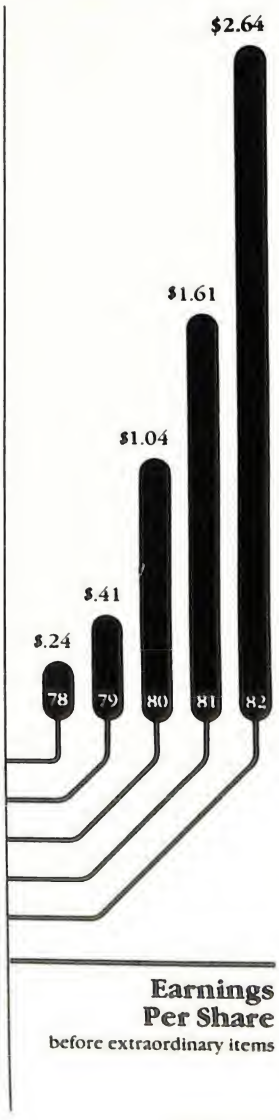
(000's omitted except per share amounts)

Year Ended 30 June	1982	1981	1980	% Change 1982 vs. 1981
Net Sales	**$304,500**	$186,500	$125,600	**+63.3%**
Gross Profit Margin	**47.8%**	44.4%	40.3%	**—**
Net Profit Margin[1]	**13.3%**	13.4%	12.9%	**—**
Net Income[1]	**$ 40,600**	$ 24,900	$ 16,200	**+63.1%**
Shareholders' Equity	**$105,900**	$ 61,600	$ 35,500	**+71.9%**
Earnings Per Share[1][2]	**$ 2.64**	$ 1.61	$ 1.04	**+64.0%**
Average Shares Outstanding[2]	**15,400**	15,460	15,593	**—**
Quarterly Earnings Per Share[1][2]	**1981-82**	1980-81	1979-80	
September 30	**$ .47**	$ .29	$ .20	
December 31	**.61**	.38	.20	
March 31	**.71**	.44	.25	
June 30	**.85**	.50	.39	
Total	**$ 2.64**	$ 1.61	$ 1.04	

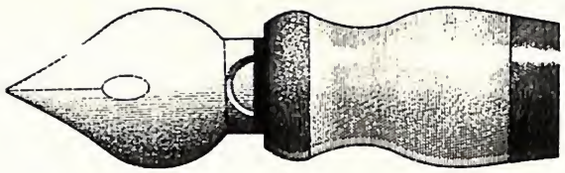
(1) Includes credit for reversal of United Kingdom taxes of $1,700,000 ($.11 per share) in fiscal 1980 and excludes extraordinary items.
(2) All per share figures adjusted for stock splits made during fiscal 1982, 1981 and 1980.







LETTERS TO THE EDITOR



Dear Editor,

I am writing on behalf of the Commodore Users Group in New Zealand. This group was formed late in 1981 and now consists of over 50 financial members, with about 120 on our mailing list. I am enclosing details of our group for your user group list in your magazine.

The New Zealand Commodore Users' Group meets on the third Wednesday of the month at the VHF Clubrooms, Hazel Ave., Mount Roskill starting at 7.30pm. Meetings usually consist of a talk on some aspect of Commodore machines, followed by workshop activities.

There are many advantages of belonging to this group including use of a group library of publications and programs, monthly newsletter, programming courses, new product demonstrations, savings on purchase of equipment and visits to computer applications.

Please contact me if you would like further information.

Yours faithfully,

ROGER ALTENA,
37 Graeme Ave.,
MANGERE EAST.
(Secretary/Treasurer)

Dear Editor,

We wish to inform you that the South Australian branch of the Commodore/VIC Computer Users Association is a functioning user group established so that enthusiasts can meet and discuss all aspects of computing.

The club meets monthly and further information can be obtained by contacting the president or secretary at the address below.

Yours faithfully,

EDDIE HANN,
Secretary.
13 Miranda Road,
PARALOWIE,
STH AUSTRALIA 5108.

Dear Editor,

I am currently using a CBM 4016 to prepare tapes for control of an N.C. Machining Centre. I was very interested in an article in Volume 2 Issue 5 of the magazine, titled 'Fast Forward On VIC Cassette".

I want to put this on to the CBM but the program when loaded would not work. Can you advise the changes required.

Regards,

J. GILLARD.

The original article was written for using the cassette with a CBM computer and was published in Volume 1 Issue 4. I have sent you a copy of this article.

ERRATA

COPY/ALL PROGRAM VOLUME 2 ISSUE 5

Page 31, Machine Language portion of program:

Line 5 sould be;

.: 0ce8 02 f0 dd 85 96 4c cc ff

Page 30, program listing

Program Line 740 should be;

740 print"[up]"

PRODUCT REVIEW

THE UMI COLLECTION

United Microware Industries are one the largest American distributors of VIC-20 software. Imagineering has begun distributing UMI software to VIC Dealers in Australia and submitted the UMI collection to the Commodore Magazine for critical comment.

Ask your Commodore Dealer for more information on VIC software.

The collection consists of seven cartridges and 12 cassettes and feature some the top talents in the business, including Len Sasso, Roger Merritt, Peter Fokos and Giguere.

CARTRIDGES

ALIEN BLITZ

– Peter Fokos and Giguere

An inplementation of the classic video arcade game Space Invaders. Nothing fancy and true to the original.

OUTWORLD

– Giguere

A classy 'shoot em down' with excellent colour, movement and control. You have to protect the space colony from falling asteroids.

SPIDERS OF MARS

– Peter Fokos

An implementation of the arcade classic Defender built around the invasion of the spiders from mars. Well programmed with fast action and good user control. This cartridge was voted game of the month by the Commodore technical department.

AMOK

– Roger Merritt

Blast your way through endless rooms full of robots. A mediocre game also available on cassttee.

RENAISSANCE

A beautifully programmed implementation of Othello. Excellent VIC graphics and the program plays a strong game.

CLOUDBURST

– *Peter Fokos*

The current program of the month in the Commodore technical department. The fastest 'shoot em up' available on the VIC-20 with enough action for the fastest fingers.

8K RAM CARTRIDGE

Just another RAM cartridge

CASSETTE TAPES

VICAT

– *Len Sasso* (unexpanded VIC)

This is a small database for VIC cassette. The program writes a block of 45 lines with 17 characters per line onto cassette. Fast forward techniques are used to access the blocks of data. This is a clever program that demonstrates advanced cassette handling.

SUB CHASE

– *Roger Merritt* (8K VIC)

You have to destroy a fleet of submarines with depth charges before you are torpedoed.

VITERM A

– *D. Lundberg* (any VIC)

A communications program similar to VICTERM. The only advantage is that this program will work with any memory configuration.

AMOK

– *Roger Merritt* (unexpanded VIC)

The cassette version of the cartridge of the 'kill the robots before they kill me' game.

SPACE DIVISION

– *D. Ballinger* (6K)

A visual delight that also teaches long division. The reinforcement sequences are complete overkill, but delightful anyway.

RACEWAY – UMI

(unexpanded VIC)

A version of the arcade classic 'Racetrack'. The game is a slow and a little cumbersome as block graphics are used.

KOSMIC KAMIKAZE – UMI

(8K)

Shoot the kosmic kamikases before they land. A slow block graphic 'shoot em up'.

ALIEN BLITZ

– *Fokos & Giguere* (unex)

The classic arcade game Space Invaders for an unexpanded VIC.

3-D MAZE – UMI

(unexpanded VIC)

You are inside a 3-D maze and must find your way out. Good VIC implementation of a game found on most other micros. The maze is changed each time the game is played.

SUPERHANGMAN

– *D. Ballinger* (unex)

Another exercise in reinforcement overkill, but a good version of hangman.

SIMON

– *D. Lundberg*(unexpanded VIC)

An implementation of SIMON for the VIC. Match sequences of tones associated with coloured bars.

VICALC

– *Len Sasso* (unexpanded VIC)

Len Sasso is one the most creative programmers working with Commodore computers. This program for the VIC-20 is an oustanding piece of software design. VICALC stands for Visible Calculator. The screen displays 4 data registers (A-D) and 10 memory registers (0-9). VICALC has a set of commands that allow you to manipulate the calculator. Operations include memory register manipulation, with clear, register add/subtract/divide /multiply/negate/zero and exchange, precision control of register contents, stack operations on the data registers and function calculations including compound interest, trig functions, square root and log functions.

Just shows you what is possible in an unexpanded VIC with a bit of imagination and clever programming.

VIC INNOVATIVE COMPUTING

by Clifford Ramshaw

This books makes available 30 excellent games including their complete listings as well as program structures. Clifford Ramshaw is recognized as one of the most creative programmers of computer games and in *VIC Innovative Computing* he has opened new dimensions in using your standard VIC 20.

Games included are: *Space Fight; Hi-Speed Maze; High-Res Draw; Warlock; Dragon's Lair; Synthesiser; Ganymede; Hoppy; City Bomber; Battleship; Maths; Duck Shoot; Grand Prix; Rat Trap; Earth Attack; Bomber Attack; Blackjack; Squash; Save The Shuttle; Hangman; Alien Overun; Invasion; Dumper; Seige; Snakes & Ladders; Dungeon; Golf; Nuclear Attack; Chess and Assassin.*

Three cassettes are also available in conjunction with the book. Each cassette contains seven games from the book as follows:

VIC Innovative Cassette 1

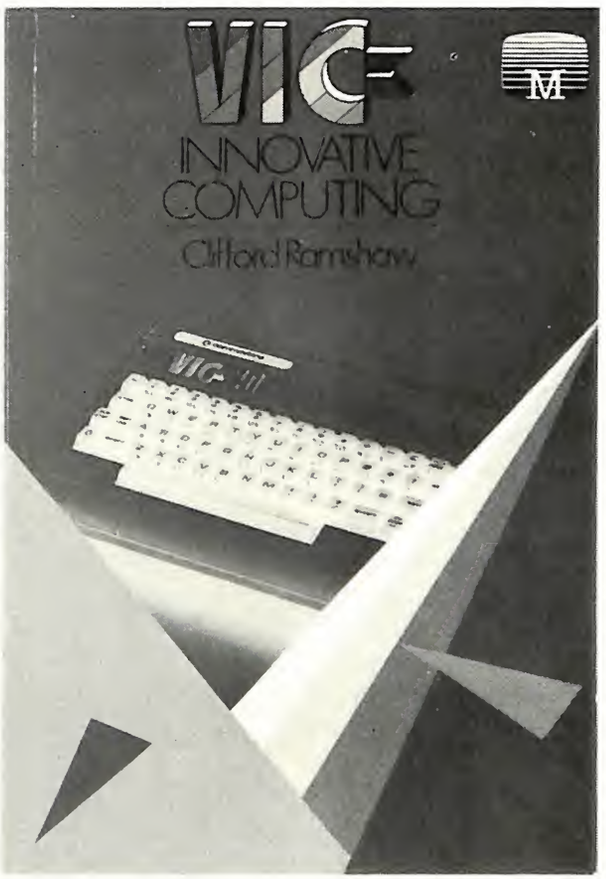
"City Bomber", "Dumper", "Nuclear Attack", "Ganymede", "Space Flight", "Battleship", "Duckshoot".

VIC Innovative Cassette 2

"Alien Overun", "Rat Trap", "Grand Prix", "Warlock", "Bomber Attack", "Hangman", "Seige".

VIC Innovative Cassette 3

"Hoppy", "Save the Shuttle", "Invasion", "Dragon's Lair", "Dungeon", "Blackjack", "Squash".



Book $17.95 + $1.50 postage & handling

Each Cassette $20.00 + $0.50 postage & handling

I would like to order the following

TITLE	COST
VIC Innovative Computing Book ($19.45)	.....
VIC Innovative Cassette 1 ($20.50)	.....
VIC Innovative Cassette 2 ($20.50)	.....
VIC Innovative Cassette 3 ($20.50)	.....
TOTAL	_____

Please mail this coupon or a copy to:

Computer Reference Guide
Suite 204, 284 Victoria Avenue,
Chatswood. NSW 2067
Phone: (02) 419 3277

Name .....

Address .....

.....

Post Code ..... Telephone ( ) .....

☐ Cheque enclosed
☐ Bankcard
☐ Diners Club
☐ American Express

Card No. .....

Expiration .....

Signature .....

VIC 20 The friendly computer

NEW VIC CARTRIDGE GAMES.

Reviewed by Charlie Green.

Four new cartridges are going to be released very shortly by Commodore, these are Omega Race, The Sky Is Falling, Mole Attack and Gorf.

Three of these games come straight off the arcade machines. And being somewhat of a connoisseur of these types of amusements, I think that some of these are an improvement on the real thing. Others like The Sky Is Falling are no-improvement games and are very similar to the games seen in pubs and clubs about 3 or 4 years ago. Gorf is something else – this game is still popular even after two and a half years in the field and so cannot miss as a VIC game.

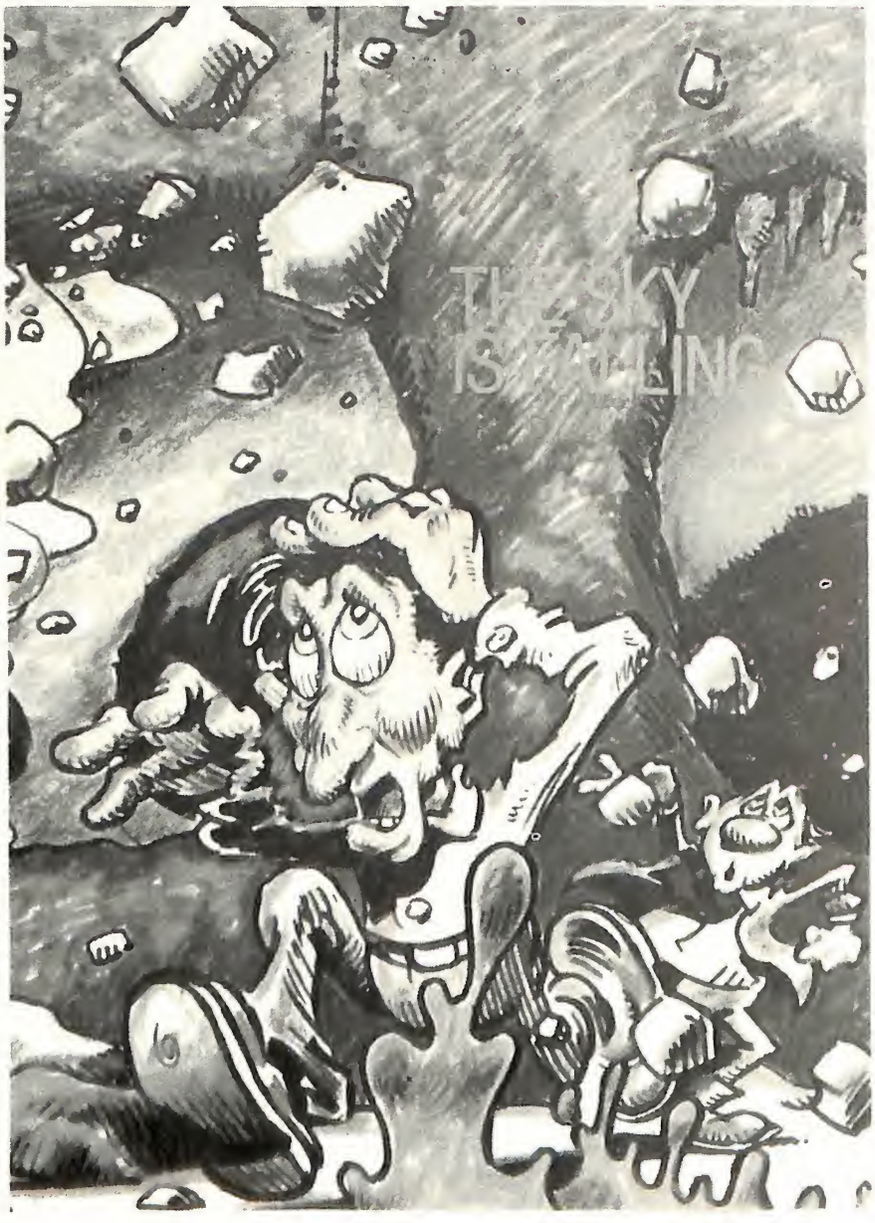
The Sky Is Falling.

Back to the paddles for this game. One or two players are allowed for in this game. On the top of the screen are displayed a number of variously sized boulders, which then drop down layer by layer. The object of the game is to stop these boulders from hitting the ground.

Supplied to do this are three bats layered one on top of the other. As the boulders come down, the bat can be manouvered to hit the rocks which then score points. The points scored relate to the level of the game and to which layer of boulders is dropping. As the layers are depleated the number of bats gets reduced and the speed of the boulders are increased.

There are two simple strategies; one – to move the bat from side to side as fast as possible; and secondly to try to follow the balls movement to optimise bat movement. Method one, although pleasing to watch – will break the paddles. Method two then takes over and seems to work well.

Generally a good fast game, but could become slightly monotonous after a month or two. Also requires the paddles to operate.



Mole Attack.

This one should inspire a few good laughs for both adults and children alike.

Basically one can get extreme satisfaction from clobbering as many moles as one can. There are nine mole holes on the screen in which either a head or tail appears. Upon hitting the mole with the mallet (large and wooden) the mole makes the appropriate noise.

Points are scored for how quickly you respond to the moles appearance. Ranging from +5 to -5 depending upon heads or tails. The mallet can be controlled either from the keyboard or from the joystick – the joystick being the easier of the two.

This one is a must for all families with children or adults.

Omega Race.

This game never really took off in the arcades – maybe because this version did not contain a joystick where as the VIC version does. This is the ultimate in Star Wars type games, it even plays the Star Wars music, or nearest to.

This is a full-resolution game, and so there are only two colours on the screen. Do not let this put you off, this game is the first game I have played where the user does not need stamina or luck, just skill. Floating in space is your star-fighter craft; as in space if you thrust you keep moving unless you thrust in the opposite direction. If you hit a barrier you bounce.

Once you have mastered the skill of movement after about two man months, you can concentrate on zapping the alien force – this being the object of the game.

There is only one way in which you would avoid buying this game, that is - ignorance of the games existance. After all, if this game can keep our Commodore staff occupied for over two months it has to be one 'WHAM' of a game!

Gorf.

In Sydney there is an amusement parlour run by a company which sells many of the expensive electronic entertainment machines. This company has had one large machine playing Gorf in the parlour for well over a year and a half. This machine has been steadily bringing in the dollars – so much so that the arcade has sprouted three other duplicate machines.

Unfortunately the VIC version of this game does not have so many 'bells and whistles' as the arcade version, such as voice synthesizers.

Gorf consists of several levels of which the ultimate aim is to reach a level where the player can shoot down the Gorfian flagship, much like another well known game.

Before getting this far into the game the user first has to play a type of Invaders game where the protection bases have been replaced with a flashing force field.

Next comes laser canons, which involves a galaxian type game, the difference is that as the aliens descend they throw down solid laser beams. If one hits these laser beams as they come down, all is lost.

After all this, if you are still alive, you can have a try at destroying the Gorfian flagship. A game not to be missed.

OPERATING THE DATA CASSETTE

by Garry Mason

As you should already know, your cassette unit can record program files and load them again. Another facility that is commonly used with cassette units, is its ability to record sequential files of data. Storing this data on the cassette in sequential format, reading, writing and modifying data is the purpose of this article.

The Commodore cassette unit is similar to the normal everyday cassette unit that we record and play music on, except that it contains a special circuit to encode and decode the digital information. This circuit is used to produce all those high pitched sounds that can be heard if a program tape is played on a normal cassette unit.

The use of physical cassette buttons on the unit to control tape direction does restrict the programmer in the use of the cassette drive, however the cassette motor can be controlled. This enables the user to start and stop the cassette operation as required. Another feature of the Commodore cassette unit is to sense whether any of the buttons on the cassette unit are actually in the down position. More about this later.

A sequential file is like a large roll of sticky tape in one of those plastic holders. To get to the end of the roll, it is necessary to unroll the whole tape first. The same can be said of the cassette file. If you have written a large file on the cassette unit and you are looking for the last record you must read all the other pieces of information in first to get to the last section of information – that being the information you want.

The OPEN Statement

The OPEN statement is used to tell the computer that we want to use a file. It also conveys such information as, what type of file we wish to use and on what device. Another thing we have to tell the computer, is what do we want to use the file for! The format for the OPEN statement is as follows:-

OPEN F, D, S, "filename"

The first value, variable 'F' is used to tell the computer which number you are going to associate with that file. For example, if you OPEN the file with 'F' set to 2, then it is the number two we use in all PRINT#, INPUT#, and GET# statements. This number can be any value between 1 and 255. One thing to remember is that if you OPEN a file with 'F' equal to or greater than 128, the PRINT# statement will then transmit a linefeed character at the end of each line regardless of terminator. 'F' is usually called the logical file number.

The second value 'D' is known as the device number. The value of this is the factor that decides whether the disk or the cassette is used in all following statements. The correct device numbers can be found by looking the number up in Table 1. The cassette drives are device numbers 1 and 2. Only device 1 in used with the VIC-20.

The secondary address, variable 'S', in the case of the cassette unit is used to tell the computer how we want to use the cassette. By using a secondary address of 1 we are telling the computer that we want to write a file. If we used a secondary address of 0 it would indicate that we are going to read from the cassette file.

The filename is only optional, both on read and write operations. Generally it is best to put a name on data so that it can be searched for. When the machine is searching for a record it is possible to use a null name, in which case the program will try to read the first file it comes across.

Device	Device Number	Secondary Address	Operation Performed
Keyboard	0	None	
Cassette Drive #*	1 (Default)	0	Open for read
		1	Open for write
Cassette Drive #2	2	2	Open for write, but add End of Table mark (EOT) on close
Video Display	3	None	
Line Printer Models 2022 and 2023	4	0	Print data exactly as received
		1	Print data using previously defined format
		2	Received format to be used in subsequent formatted printout
		3	Receive lines per page specification
		4	Enable printer diagnostic message
		5	Create a special character
		6	Set spacing between lines (Model 2022 only)
Disk Drives (all models)	8	0	Load a program file to the computer
		1	Save a program file from the computer
		2-14	Unassigned
		15	Open command/status channel
Other devices connected to IEEE 488 Bus	5,6,7 and 9 through 31		Device numbers and secondary addresses are selected and assigned by the manufacturer of the device connecting to the IEEE 488 Bus
	32 to 255 unavailable at this time		

* This is the cassette drive mounted

The INPUT# Statement

This is the actual statement that brings the information from the cassette into memory. The way in which it does this is quite interesting. Firstly when the cassette unit is initialised via the OPEN statement, the header block is searched for and read in. This block tells the computer what type of file it has just found and any particular pieces of information required for processing the file. Nothing more is done until the first INPUT# or GET# statement is found in the program. When one is found, the cassette motor starts again and reads in the first 191 characters into a buffer area of memory. Once this is done the computer will assign the data in the buffer into variables which were given in the INPUT# or GET# statement.

The way in which this is done is also quite important, because it is easy to forget when making the file on the tape – which could result in an unreadable file. Or at least only readable via the GET# statment.

When the computer is looking to find a variable in the buffer, it starts at the present position in the buffer. Consider to following :-

```
100 INPUT#1, A$, N, B$
```

The INPUT# statement will fill up with 'A$' all the characters it can find in the buffer up to the first terminator character. Then after the terminator it will search for a number to fit into 'N'. With Commodore computer terminator characters are the comma CHR$(44) and the carriage return character, CHR$(13). There is another type of terminator, this is the absence of one. If there are no terminators on a file, then the input process will stop when the computer tries to put more than 255 characters in the string variable. In this case the program will be stopped with a STRING TOO LONG error.

Numbers also need to be seperated from other text by the use of terminator characters. The other common error condition is one caused by the data on the file being written on the file in a different order to which it is being read.

In the previous example, if the input statement were to try and input the following data sequence, it would fail with FILE DATA error;

"ABCD[cr]ABCD[cr]ABCD[cr]".

This occurs when the INPUT# statement finds a string while attempting to read a numeric value. In other words it does not fit.

Remembering these points, it is possible to design data on the tape that will not cause any problems. In other words, by including terminators in the right place, to indicate new variables and recording numerics as strings just to be on the safe side.

The GET# Statement

This is much like the INPUT# statement, only its purpose is to input a single character from the device specified by the logical file number. As it is almost impossible to tell what type of data will be read, it is normal to use a string variable. This is to stop instances where the computer will stop execution due to a SYNTAX ERROR, as it is this error that is reported when trying to put an alphanumeric value into a numeric value.

The use of the GET# statement only really comes into its own when it is used with files of fixed length. In this type of setup, the file would always be written out with the same length, regardless of the length of the actual data. This is done by filling up the rest of the data field with spaces. This then gives us records of fixed length.

Coming back to the use of the fixed length field with the GET# statement; being able to predict the length if the file becomes an asset. Using the fixed length allows us to use the GET# statement to read in the whole file and then segment the final record into several data elements. This is the theory behind the following pair of programs (Program 1 and Program 2).

PROGRAM 1.

```
 5 OPEN 1,1,1,"TESTFILE"
10 FOR C=1 TO 4
20 INPUT A$
30 A=LEFT$(A$+"          ",10) :
                    REM 10 SPACES
40 PRINT#1,A$;
50 NEXT C
60 PRINT#1,CHR$(13);
70 CLOSE 1
```

PROGRAM 2

```
10 OPEN 1,1,0,"TESTFILE"
20 FOR C=1 TO 30:GET#1,B$:
          A$=A$+B$:NEXT C
30 CLOSE 1
40 B$=LEFT$(A$,10)
50 C$=MID$(A$,10,10)
60 D$=RIGHT$(A$,10)
70 PRINT B$, C$, D$
```

The CMD Statement

The operation of the CMD or Command Statement is quite simple, in effect what it does is to transfer all normal screen output to the device associated with the logical file number. Once the CMD has been given, its operation can be cancelled by executing a PRINT# statement.

As you probably know the main use of the CMD statement is to obtain listings of programs. It is almost never used in connection with files because it can be unpredictable.

The PRINT# Statement

Through the use of this statement, we can put data into the file.

Generally the PRINT# statement works in the same way as the PRINT except for the printing of numerics. Compare the results obtained from the following program lines.

```
A=15
PRINT CHR$(34);A
OPEN 1,3:PRINT#1,CHR$(34);A:
                    CLOSE 1
```

The difference is that when PRINTing a variable normally, a space is PRINTed in front of the number and a cursor-left behind. When doing the same thing but printing through a channel number, it produces a space both in front and behind the actual variable.

This is not very important really but it does show that we need to be very careful.

Commas and semicolons are also very important, for example, if you are separating variables and the PRINT# is used – with commas you will find that the size of the file will be greatly increased as the line will be spaced out with space characters. Semicolons

are used instead to prevent this and group the data together.

When the computer executes a PRINT# statement and it gets to the end of the variable list, it will automatically send a carriage return character to the file. This can be stopped by putting a semicolon at the end of the line. If we remember that CHR$(13) or carriage return is one of the two possible terminator characters, we can put this to good use.

For example:

```
10 CR$=CHR$(13)
20 PRINT#1, A$; CR$; B$
30 PRINT#1, C$
40 PRINT#1, D$
```

Unfortunately the above program lines would not work if one was writing to a record in a relative file on the disk unit. This is because of the way in which the system keeps track of the record number. What happens is that the system treats each PRINT# statement as contining one record. So the above program lines would infact write out three different records; one after the other.

Writing Cassette Files

The three statements that have to be used to get data onto the cassette unit are the OPEN, the PRINT and the CLOSE statements. This also happens to be the order in which they need to be executed. The OPEN statement prepares the cassette unit and writes the header block. The PRINT# statement puts variables or literals into the cassette buffer area. All cassette operations are done via the 191 byte buffer area – this buffer only gets written to the tape when the buffer gets full. And finally the CLOSE statement that writes an end-of-file marker on the cassette and then frees up that logical file number.

As mentioned, all cassette operations are done via the buffer area. Data is written from the buffer area; not directly from the PRINT# statement. As the buffer is 191 bytes long, the system can afford to wait until the buffer is full before it actually does anything to the tape. The buffer is only written out when the 192nd character is placed in the buffer. When this happens the first 191 characters are written on the tape and the 192nd now becomes the first character waiting to be written.

For this reason it is sometimes very convenient to write data out in blocks of 191 (including terminator characters). This then makes reading back data more efficient. As data is read in blocks of 191 it would be silly to read in two blocks if only 192 or 193 characters were required.

The following two programs demonstrate how this technique can be used. Basically what the programs do is to write onto the tape complete buffers (191 bytes) – even if the buffer is not full and then read the information back in, a buffer at a time.

PROGRAM 1

```
10 REM PROGRAM TO WRITE
DATA ONTO TAPE IN BLOCKS
                    OF 191 BYTES
20 OPEN 1,1,1, "TEST FILE"
30 FOR C=1 TO 5
40 PRINT#1,"1234";CHR$(13);
            "FRED";CHR$(13);
50 GOSUB 100
60 NEXT C
70 CLOSE1 : END
100 REM WRITE OUT BUFFER
                    ONTO TAPE
110 POKE 166, 190 : REM SET
          POINTER TO LIMIT - 1
120 PRINT#1," "; : REM PUT 192
          NO CHAR IN BUFFER
130 POKE 166,0 : REM RESET
      POINTER FOR MORE DATA
```

PROGRAM 2

```
10 REM PROGRAM TO READ
TWO STRING VARIABLES
    STORED IN BLOCK FORMAT
20 OPEN 1,1,0
30 INPUT#1, A$, B$ :
REM READ DATA FROM BUFFER
40 PRINT A$, B$
50 GOSUB 100 : REM CHECK
                FOR END-OF-FILE
60 GOSUB 200 : REM READ NEXT
                    BLOCK IN
70 GOTO 30
80 REM
100 REM CHECK FOR
                    END-OF-FILE
110 IF ST<>0 THEN CLOSE 1 : END
120 RETURN
200 REM READ IN NEXT BUFFER
                    / BLOCK
210 POKE 166, 190: REM SET
                POINTER TO LIMIT
220 GET #1, A$ : REM FORCE
        NEW BLOCK TO BE READ
230 POKE 166,0 : REM RESET
      POINTER READY FOR READ
240 RETURN
```

These programs really never needed to be that complicated, just to store two strings of a file. But they do show the steps that need to be gone through to do the task all that is required is the following:-

PROGRAM 1 Summary

1. OPEN the file with secondary address set to '1' to indicate we wish to write the file (line 20).
2. The use of terminator characters to separate data on the file and the semicolon to avoid spaces or gaps in the data. The use of the ; on the end to prevent two terminators being written onto the tape (line 40).
3. CLOSEing the file to clear out the buffer and write the End-of-file marker (line 70).

PROGRAM 2 Summary

1. OPEN the file for read in line 20.
2. The use of the INPUT# statement to bring the data from tape to buffer and from buffer to variable storage (line 30).
3. The check to see if we have found or read the End-of-file marker by the use of the status variable 'ST' (line 110).
4. CLOSEing the file and freeing up logical file 1 (also line 110).

Reading Files Back

The rule here is very simple, you read in data exactly the same way it was was written. If you wrote a string, read in a string variable; if you wrote out a numeric then read in a numeric. And if you are not sure – what is the use of the data anyway (the GET# statement can be used to retreave the information with these types of files).

////example////

EXAMPLES

(the variable CR$=CHR$(13)= the terminator character).

PRINT# Statement	INPUT# Statement
PRINT#1,A$;CR$; B$;CR$;C$;CR$;	INPUT#1,A$,B$,C$
PRINT#1,A$;CR$; B$;C$;CR$;	INPUT#1,A$,B$
PRINT#1,A;CR$;B; CR$;C;CR$	INPUT#1,A,B,C
PRINT#1,A;CR$;B; C;CR$	INPUT#1,A,B Note B is not the total of B + C, but is the concatenated total.
PRINT#1,A$;CR$; B$;CR$;A;CR$; C$;CR$	INPUT#1,A$,B$,A,C$

Modifying A Cassette File

This is one of the most difficult things to do with any sequential file system. The process is not too bad in a system which can have two tape units. The Computer can use one to read the old file and on another to write the modified records onto.

For people without two cassette units or VIC-20 owners (the VIC only supports one cassette unit) life is more complcated. There are two commonly used methods – one for people with lots of memory and one (which is more complicated) for those with limited memory.

Method 1 – for those with lots of memory.

Step 1 Read all of the old file into memory.
Step 2 Modify any records that need amending.
Step 3 Re-wind the cassette.
Step 4 Write out all of the file back onto the tape.

Method 2 – for those people with limited memory.

This method requies that large files are split up into several small files positioned at set spaces along the tape. The tape counter can be used for this. Or one can use the utility for computer–controlled tape movement (VIC Issue 5 Vol. 2, CBM Issue 4 Vol. 1).

Step 1 Rewind tape to start of tape.
Step 2 Move tape to start of the first file.
Step 3 Read in all of file.
Step 4 Modify all or any records in block.
Step 5 Rewind tape to the start of the tape.
Step 6 Move out to the start of the same file.
Step 7 Write out the whole of the file again, over-writing the original file.
Step 8 Rewind the tape to start again.
Step 9 Move out to the next small block file.
Step 10 Go back to Step 3 and continue until all the small-block-files have been changed.

After all that the job should be finished. If you do not like the sound of programming all that you may like to look at a program called VICat. This is available from Imagineering and their distributers.

The 'ST' Status Variable

This variable is updated after every I/O (Input/Output) statement and reflects the present status of the last device used. This can be used to detect the End-of-file marker as on line 110 in Program 2. Which will be set to either 64 or -128 depending upon which marker was written when the file was CLOSEd.

End-of-file or End-of-tape Markers.

When files are OPENed with a secondary address of 1, an End-of-file marker will be written onto the tape when the CLOSE statement is executed. What happens is that the system writes a CHR$(0) onto the end of the tape, either by putting CHR$(0) into the remains of the present buffer or by writing a new buffer onto the tape. When this is picked up via a read the value of 'ST' is changed to 64.

If an End-of-tape is encountered 'ST' will be changed to -128. This is done by OPENing the file with a secondary address of 2. That is really all that happens. The only other thing that happens when an End-of-tape is found, is during searches – they are stopped upon finding these markers.

Status Byte Returned by External Devices Via Variable ST

Device Operation	Status							
	00000001 Read as 1	00000010 Read as 2	00000100 Read as 4	00001000 Read as 8	00010000 Read as 16	00100000 Read as 32	01000000 Read as 64	10000000 Read as –128
Read from Cassette drive #1 or #2	Operation OK	Operation OK	Short Block Data block read had fewer bytes than expected	Long Block Data block read had more bytes than expected	Unrecoverable read error	Checksum error. One or more data bits read incorrectly	End of file encountered	End of tape encountered
Verify cassette drive #1 or #2					Any verify mismatch		None	
Disk drives (all models)	Receiving device not available	Transmitting device not available	None	None	None	None	End of file	Disk drive not present

Review of the A.S.K. software

A.S.K. is based in London and produces educational computer programs designed to complement work at school. These tapes are high quality educational programs that are aimed at the six to eleven age group. They are cassette loaded into a VIC which requires a 16K RAM pack.

All these programs are now available from your Commodore Dealer.

Twister

Is a geometric puzzle involving coloured squares arranged in columns.

The person trying to solve Twister must first decide how many columns to use. A number between 3 and 10 is allowed and the higher the number, the harder the puzzle!

If, for example, 4 columns are selected, you start with a block of coloured squares arranged with all the colours in columns. The object of the game is to rearrange the blocks so that no column or row contains a repeated colour.

Rainbow Towers

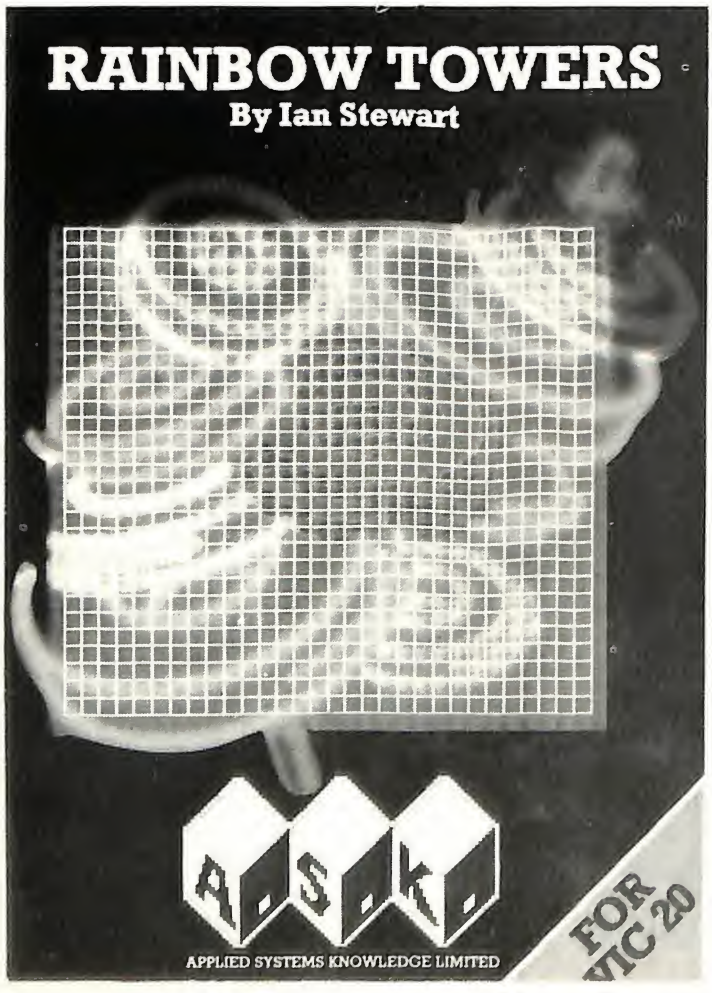
Is a computer variation on the famous 'Towers of Hanoi' puzzle. In 'Towers of Hanoi' you have three towers and on one of them are three different sized discs. The aim is to transfer the three discs to another tower. To do so, you move the discs from tower to tower, and at no time may a disc be placed on top of a smaller one.

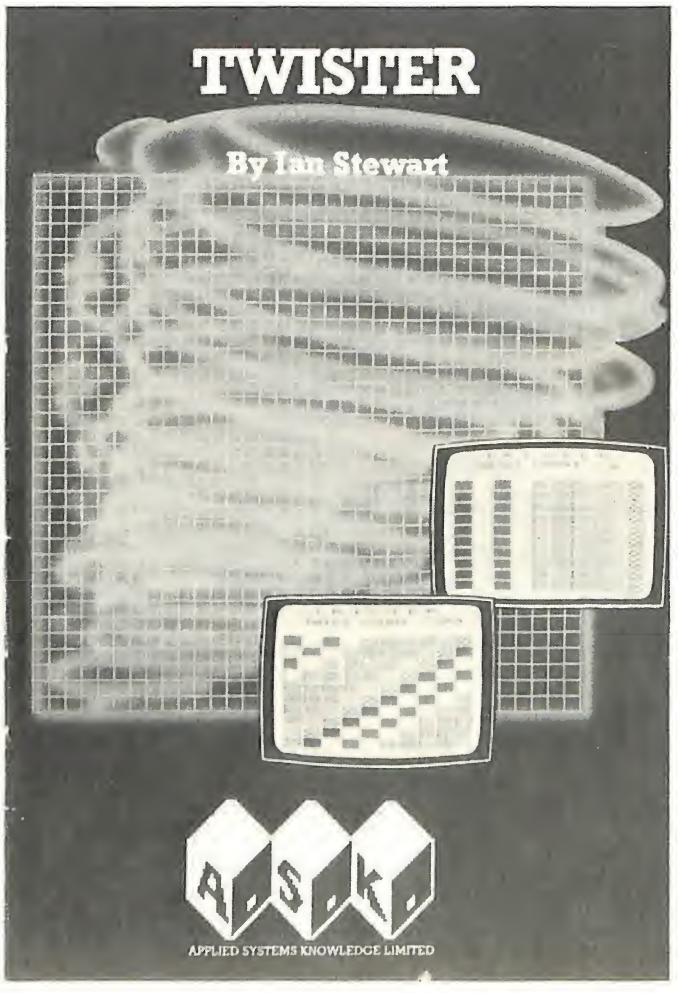
In Rainbow Towers, there a six towers. Three of these contain coloured discs and you have to move them to the other towers. However, as the discs move they change colour. The puzzle is only solved if the three towers which are empty at the beginning end up with discs of the right colour.

We Want To Count

Introduces young children to simple counting and the number 1 to 5. An adult or older child is needed to help get the program loaded, to select which of the 4 different episodes the young learner is to work on, and to explain what to do.

Thereafter, parent and child can work together or if the child is





confident enough, they can be left to get on with it so long as help is close at hand.

Only two of the keys on the VIC keyboard are used by the child. They are the space bar, the biggest key of all, and the RETURN key. The child isn't expected to be able to read the word RETURN which is printed on the key, but with the help of an adult or older child will soon remember where it is. If another key is pressed by accident it can do no harm as none of the other keys will work whilst the program is being run.

Facemaker

Is for children, but adults will also find it both fascinating and enjoyable. Very young children may need help to read the instructions and type in their choices.

You must choose the name and features of the person to be drawn. Step by step the face builds up, appearing on the screen at the touch of a button.

To make Facemaker work, you must:

READ the question; e.g. WHAT ARE MARY'S EYES LIKE?
READ the response options; e.g. LARGE OR SMALL
SELECT the response; e.g. LARGE
CORRECTLY TYPE the word chosen; e.g. L-A-R-G-E

The completed picture prompts questions about the personality behind the face. Does the person look kind, friendly or bad-tempered? Perhaps the face was supposed to look like someone in particular -does it? What is different?

Number Chaser

Can be used by any child who knows how to multiply two numbers together. Various levels of difficulty are included in the program so that children of all abilities will enjoy it and benefit from using it.

The program is designed to give children practise in estimation, a skill that has become an important part of everybody life. For example, every time a calculator is used, it is essential to be able to decide whether the answer given is approximately right or not. Blind faith in the calculator's answers could lead to serious losses in a commercial environment. On the factory floor, deciding whether to stop production because a fault has developed in a particular machine, may depend on an estimate of the effect it will have on the finished product.

In Number Chaser, an exciting car race provides an opportunity to practise and improve your estimating skills.

Number Gulper

Gives children practice in all the four basic arithmetic operations -addition, subtraction, multiplication and division. For example, the quicker a child can recall or work out what seven times nine is, or five times eight or ..., the better that child will be at Number Gulper. And because it is such an exciting and motivating game to play, children will want to try again and again.

The player is asked to make a number using only those numbers diplayed on the screen. Each of these numbers has a colour associated with it indicating which operation (add, subtract, multiply or divide) that number can be used for. Success with Number Gulper depends on choosing the method which involves collecting the fewest coloured numbers.

Later on, hazards appear into which the "gulper" sent round to collect the numbers for the player may disappear, never to be seen again. A change of strategy may be necessary as a player has only three gulpers for each game.

PROGRAMMERS' AID CARTRIDGE

by Brad Fisher

The VIC 1212 Programmers' Aid Cartridge assists new and experienced programmers in writing and editing programs. It adds special editing commands to the VIC's vocabulary and also makes some use of the programmable function keys.

The cartridge is activated with a SYS call, SYS 28681. A message is displayed on the screen to indicate that Programmers' Aid is in operation.

Commands handled by the cartridge are:

AUTO	CHANGE	DELETE	DUMP
EDIT	FIND	HELP	KEY
KILL	MERGE	OFF	OFF
PROG	RENUMBER	STEP	TRACE

★ AUTO provides automatic line numbering. A command such as AUTO 1000,5 will prompt at the beginning of each line with a line number starting with 1000 and incrementing by 5 after each line is entered.

★ CHANGE allows keywords or text to be changed. For example, CHANGE GOTO,GOSUB will change all occurrences of the keyword GOTO into GOSUBs. CHANGE B$,A$ will change all occurrences of the variable B$ into A$. CHANGE 'HELLO' , 'GOODBYE' , 200-400 will change HELLO into GOODBYE in the range lines 200-400. However, if you try to change the variable A into B, variables such as AZ or A$ will also be changed into BZ and B$. Any A in a REM statement would also be changed to a B. I'd call that a bug -Commodore would call that a feature!

★ DELETE will delete a series of lines. No more cramps from having to type 300 line numbers followed by returns! It works to the same format as LIST: DELETE 100, DELETE-100, DELETE 200-290 etc.

★ DUMP will display the current value of all simple variables, that is, variables like A, AB, SY%, FI$. It should have been a simple matter to display array variables like A$(4), BF$(19,6) etc. but regrettably it has been neglected.

★ FIND GOTO will search your BASIC program and display the lines in which GOTO appears. A line number range may be specified. If the code or text occurs more than once in the line, the same line is printed again. I suppose it's the lesser of two evils – you might miss the second one if the line is printed only once.

★ HELP is quite handy. Ever felt like tearing your hair out because the VIC could only wink at you under a SYNTAX ERROR message? Type in HELP and the VIC will show you the approximate place where it fell over by highlighting it in reverse field. I say approximate because a comma may be highlighted when the PRINT# command preceding it may be the culprit. I think it's one command that should have been included in the operating system of the VIC to start with.

★ MERGE is useful if you have a library of special routines on cassette. MERGE 'FILENAME' will load the program into memory – without destroying any program already there. Be careful, though. The merged program will overwrite any identical line numbers in the original. One good reason to reserve line numbers for specific routines, but that is the subject of another article.

★ RENUMBER is self explanatory. RENUMBER starting line, increment. I only wish you could specify a range of line numbers to be renumbered. As it is you must renumber the entire program. If you wish to ruin a program, try RENUMBERING with an increment of zero. It should run perfectly unless you use GOTO or GOSUB!

★ TRACE will print the line number currently being executed in the top right corner of the screen in a window with the previous 5 lines executed above it. The manual states that this command tends to slow down the execution speed of the program, and Boy! they're not kidding! It does get confused, however, if you have renumbered the program to be all the same line number. The speed can be slowed down (even further?) by holding the CRTL key, the shift key or pressing SHIFTLOCK.

★ STEP works the same way as TRACE but will execute only when CTRL or SHIFT is held down. To cancel either STEP mode or TRACE mode, type OFF.

★ KILL turns off the most of the cartridge's functions. This is because BASIC runs more slowly when the cartridge is used. The programmable function keys are used by the cartridge in a minor way. By pressing one either normally, shifted, or when holding CTRL, up to 10 characters are entered as though from the keyboard. These have preassigned values at first, but may be redefined by the user by use of the KEY command. An

The VIC responds by immediately performing the calculation . . . and PRINTs the answer . . . 10. Note that you DON'T USE QUOTATION MARKS when using the PRINT statement to do calculations. Try these two examples to see why:

PRINT "6+4" (hit RETURN) The VIC displays 6+4 instead of 10.

PRINT "The sum of 6+4 is" 6+4 (RETURN). VIC CALCULATES Numbers OUTSIDE the quotes, but PRINTs the numbers and words INSIDE the quotes. This is a useful technique to remember.

Mathematical Calculation—Examples & Notes

Addition
PRINT500+1000

Subtraction
PRINT6-4

Multiplication
PRINT5*5

Computer multiplication uses the * (asterisk) instead of the X (times) sign to avoid confusion between the multiplication sign, the letter x, and the graphic symbol X. Just remember that when you are using your VIC to multiply numbers, the asterisk (*) is used instead of the (X) times sign.

Division
PRINT10/5

Division on the VIC uses the fraction symbol (/) instead of the division sign (÷). To divide 10 by 5, type 10/5 instead of the conventional 10÷5. The VIC gives you the answer 2.

Fractions
PRINT5/10*10/5

Fractions are handled like normal fractions. 5/10 multiplied times 10/5 equals the whole number 1 and that's the answer the VIC gives if you type in this example.

Decimals
PRINT5.2/5.8

The answer to 5.2 divided by 5.8 is .896551724. The VIC handles decimal answers in the range from 0.01 to 999,999,999 in standard notation, but numbers beyond this range are automatically converted to *scientific notation*. This means that if a number is too large to conveniently display or work with, it is converted to a shorter "scientific" form (see below).

Negative Numbers
PRINT-5*10 . . . or . . . PRINT-5*-10

The VIC recognizes and handles negative numbers, but complex formulas may require parentheses to keep negative numbers from being confused with subtraction operations. For example, 5 might be typed as (-5) if there are many operations in a single calculation.

Exponents
PRINT2↑2

This means PRINT 2 to the 2nd power, normally written as 2^2 (or 2 times 2). The VIC uses the up-arrow to show this. 10 to the 3rd power (10 times 10 times 10) is written in VIC terms as 10↑3 . . . (1000). If you're not familiar with exponentiation, 10↑3 means takes the first number (10) and multiply it times itself 3 times. 5↑2 means multiply 5 times 5 (or 25). Exponentiation is important because it is used by the VIC to designate numbers that are too large to express in normal form.

Using Pi

One of the VIC's special characters is the "pi" symbol. Pi looks like this: π, and is located on the **front of the up-arrow key.** You can print or display the pi symbol like any other character, but pi can also be used as a **value** like a number. The value of pi is 3.14159256 . . . etc. and represents the ratio of the circumference of the circle to its diameter. If you multiply the diameter of a circle by pi you will get the circumference. To use the VIC's pi character, simply hold down the SHIFT key and press the pi key. Try typing the following:

PRINTπ

The VIC responds by displaying the number 3.14159265. Now let's say you wanted to calculate the circumference of a circle with a diameter measuring 5 inches. Simply type: PRINTπ*5 and hit RETURN. The VIC answers 15.7079633. Likewise, you can find the diameter of a circle by dividing a circumference by pi.

The Order in Which VIC Calculates

If you are using a complex formula or several mathematical operations in a long calculation, the VIC will always calculate in the same order . . . according to the **type of operation.** The order in which the VIC examines a calculation or formula is: 1) exponents, 2) multiplication and division, 3) addition and subtraction. If there are several calculations in the *same category*, the VIC performs them starting **from left to right.**

This is especially important if you are using the VIC to solve equations and perform multi-operation calculations or formulas.

You can force the VIC to calculate in the order you want by using parentheses ( ) to isolate operations you want performed separately, in which case the VIC calculates starting with the operation in the **innermost parentheses.**

Let's look at an example. Type in the following and hit RETURN:
PRINT10*2↑2/4+3-2

In this formula, the VIC will perform the calculation in the following order: first the exponentation is considered. Next comes multiplication and division, followed by addition and subtraction. If there are several operations of the same

The VIC responds by immediately performing the calculation . . . and PRINTs the answer . . . 10. Note that you DON'T USE QUOTATION MARKS when using the PRINT statement to do calculations. Try these two examples to see why:

PRINT "6+4" (hit RETURN) The VIC displays 6+4 instead of 10.

PRINT "The sum of 6+4 is" 6+4 (RETURN). VIC CALCULATES Numbers OUTSIDE the quotes, but PRINTs the numbers and words INSIDE the quotes. This is a useful technique to remember.

Mathematical Calculation—Examples & Notes

Addition
PRINT500+1000

Subtraction
PRINT6−4

Multiplication
PRINT5*5

Computer multiplication uses the * (asterisk) instead of the X (times) sign to avoid confusion between the multiplication sign, the letter x, and the graphic symbol X. Just remember that when you are using your VIC to multiply numbers, the asterisk (*) is used instead of the (X) times sign.

Division
PRINT10/5

Division on the VIC uses the fraction symbol (/) instead of the division sign (÷). To divide 10 by 5, type 10/5 instead of the conventional 10÷5. The VIC gives you the answer 2.

Fractions
PRINT5/10*10/5

Fractions are handled like normal fractions. 5/10 multiplied times 10/5 equals the whole number 1 and that's the answer the VIC gives if you type in this example.

Decimals
PRINT5.2/5.8

The answer to 5.2 divided by 5.8 is .896551724. The VIC handles decimal answers in the range from 0.01 to 999,999,999 in standard notation, but numbers beyond this range are automatically converted to *scientific notation*. This means that if a number is too large to conveniently display or work with, it is converted to a shorter "scientific" form (see below).

Negative Numbers
PRINT−5*10 . . . or . . . PRINT−5*−10

The VIC recognizes and handles negative numbers, but complex formulas may require parentheses to keep negative numbers from being confused with subtraction operations. For example, −5 might be typed as (−5) if there are many operations in a single calculation.

Exponents
PRINT2↑2

This means PRINT 2 to the 2nd power, normally written as 2^2 (or 2 times 2). The VIC uses the up-arrow to show this. 10 to the 3rd power (10 times 10 times 10) is written in VIC terms as 10↑3 . . . (1000). If you're not familiar with exponentiation, 10↑3 means takes the first number (10) and multiply it times itself 3 times. 5↑2 means multiply 5 times 5 (or 25). Exponentiation is important because it is used by the VIC to designate numbers that are too large to express in normal form.

Using Pi

One of the VIC's special characters is the "pi" symbol. Pi looks like this: π, and is located on the **front of the up-arrow key.** You can print or display the pi symbol like any other character, but pi can also be used as a **value** like a number. The value of pi is 3.14159256 . . . etc. and represents the ratio of the circumference of the circle to its diameter. If you multiply the diameter of a circle by pi you will get the circumference. To use the VIC's pi character, simply hold down the SHIFT key and press the pi key. Try typing the following:

PRINTπ

The VIC responds by displaying the number 3.14159265. Now let's say you wanted to calculate the circumference of a circle with a diameter measuring 5 inches. Simply type: PRINTπ*5 and hit RETURN. The VIC answers 15.7079633. Likewise, you can find the diameter of a circle by dividing a circumference by pi.

The Order in Which VIC Calculates

If you are using a complex formula or several mathematical operations in a long calculation, the VIC will always calculate in the same order . . . according to the **type of operation.** The order in which the VIC examines a calculation or formula is: 1) exponents, 2) multiplication and division, 3) addition and subtraction. If there are several calculations in the *same category*, the VIC performs them starting **from left to right.**

This is especially important if you are using the VIC to solve equations and perform multi-operation calculations or formulas.

You can force the VIC to calculate in the order you want by using parentheses ( ) to isolate operations you want performed separately, in which case the VIC calculates starting with the operation in the **innermost parentheses.**

Let's look at an example. Type in the following and hit RETURN:
PRINT10*2 ↑ 2/4+3−2

In this formula, the VIC will perform the calculation in the following order: first the exponentation is considered. Next comes multiplication and division, followed by addition and subtraction. If there are several operations of the same

by saying X=10. We can create other variables, too. Type this:

Y=2 (and hit RETURN)

Now we've defined TWO VARIABLES. X=10 and Y=2. The **power** of these variables is easily demonstrated. Type this:

PRINTX*Y (and hit RETURN)

The VIC multiplies the value of X (10) times the value of Y (2) and displays the answer, which is 20. In addition to direct calculation, you can design all sorts of calculator programs using numeric variables. Here's a short program that lets you enter two numbers to be multiplied and gives the answer:

10INPUTX:INPUTY:PRINTX*Y:GOTO10

Delay Loops Use Numeric Variables

We've discussed **time delay loops** in previous articles but how exactly does a "delay loop" work? It all has to do with numeric variables.

If you're programming a delay into your program, you may not know it but you're actually using a numeric variable. You see, you can specify that X equals a **range** of numbers—instead of a single number like 10. In a delay loop, you specify a variable like X as a range . . . the following example illustrates how:

10 PRINTCHR$(181); (and hit RETURN)
20 FORX=1TO200:NEXT (and hit RETURN)
30 GOTO10 (and hit RETURN)

Type RUN and the screen slowly fills up with graphic bars. Want the bars to move faster? Hold down the RUN/STOP key and press RESTORE. Now type LIST and hit RETURN. In line 20, change the number 200 to 50 and RUN the program again. The speed with which the bars are printed picks up considerably.

The **reason** a delay loop works is because we have defined the variable X as a **range** of numbers from 1 to 200. We then told the VIC to start **counting** from 1 to 200 before going to the next line. When we change the number 200 to 50, we shorten the loop by redefining X as a range of numbers from 1 to 50. The VIC counts faster, and the program moves faster.

In other words, we have a **loop** which PRINTs a character (CHR$(181) is the same as "["), counts to 200, then goes back around to line 10 and prints the character again. The semicolon in line 10 makes the VIC display the next character immediately next to the previous one. If we didn't have the semicolon all the characters would appear in a vertical column.

Defining a variable as a **range** of numbers has other uses, too. Say we want to PRINT a title and four lines of blank spaces at the top of the screen. That makes 88 spaces, right? (4x22 columns) Our program should tell the VIC to:

1. Clear the screen

2. PRINT a title at the top of the screen
3. PRINT 88 cyan spaces next to each other

Here's the program that does this:

```
10 PRINT"CYAN SPACES"
20 FORX=1TO88STEP1:PRINT"▶R";:NEXT
```

The first graphic symbol in line 20 means you've pushed CTRL/CYAN. The second symbol means you've pushed CTRL/RVS ON

The secret is the use of the loop in line 20. First we set up a number range of X=1 to 88. Then we STEP one at a time through that range. Each time we STEP we PRINT a cyan space. The semicolon says to print the spaces immediately next to each other. The NEXT tells the VIC to keep going through the loop until it hits the 88th time (end of the range we created for the variable X).

Here's a similar example using sound:

```
10 X=36876:V=36878:POKEV, 15
20 FORS=128TO254STEP1:POKE X,S:NEXT
```

Here we use two variables instead of one. X is our sound speaker, V is volume. The range of X is the range of tone values we want to use (see table of musical notes in owner's manual). We are STEPping 1 at a time to achieve a slower effect. What actually happens is that the VIC POKEs 36876,128 . . . then POKEs 36876,129 . . . then 36876,130 . . . and so on until we reach the upper limit of the range, which is 254. We could just as easily reverse the range and STEP-1 from 254 to 128.

To make this progression of musical notes into a sound effect, we just speed up how fast we STEP through the range of notes. Do this by LISTing the program and changing the STEP 1 in line 20 to STEP 10. Another alternative is to shorten the range of note values. In line 20, try keeping the STEP 1 but change the low value from 128 to 170. The higher value makes a higher sound. The shorter range means fewer notes and a faster effect.

Using X and X+1

One of the most frequent uses of numeric variables involves defining a variable at the beginning of the program, then **changing** the value of that variable later on. Often this takes the form of defining a variable like X, then adding 1 to that value to increase it (X=X+1).

The following counting program illustrates the use of a variable X and X+1, as explained in the comments below:

Program	Comments
10 X=1	Define X as the value 1
20 PRINTX	Print the value of X, which is 1
30 X=X+1	Increase the value of X by 1 (now 2)
40 GOTO20	Go back to PRINT the NEW VALUE OF X (2)

The counting program defines X as 1, PRINTS the starting value (1), then **changes** the value X by adding 1, and PRINTS the new value, then loops back each time to increase X by one, and PRINT the new value, etc.

If you want to see this program work more slowly, try adding a delay loop by changing line 40 to read: 40FORT=1TO100:NEXT:GOTO20

Delay loops like this can often be added anywhere in a program, by adding a colon and the FOR . . . NEXT loop.

Using the DEF FN Statement

In addition to its standard calculation abilities, the VIC also has a very special BASIC statement called "DEFINE FUNCTION," which lets you define a predetermined mathematical or scientific formula, and plug numbers into that formula during your program.

It helps to know something about variables before using the DEF FN, but this is not absolutely necessary. Pay close attention to the examples and experiment with a few of your own and you should catch on quickly to how this mathematical statement is used.

Let's begin with a simple example. Type NEW and hit RETURN to erase any old programs and type in this program:

Program	Explanation
10 DEFFNA(X)=.10*(X)	(RETURN) Sets up the formula.
20 INPUTX	(RETURN) Sets up your number input.
30 PRINTFNA(X)	(RETURN) Prints the formula answer.
40 GOTO10	(RETURN) Goes back to repeat line 10.

Type RUN and hit RETURN to start. (Hold down RUN/STOP and RESTORE to exit the program). This program calculates 10 percent of any number you type. When the question mark appears, type a number and hit RETURN. The VIC calculates 10 percent of that number and displays the answer.

Setting up a DEF FN Statement

Let's use our example to show you how the DEF FN statement works. To begin with, the format is always similar:

```
DEFFNA(X)=(formula)(X)
```

The DEFFN part of the line is simply the DEFine FuNction statement.

The A following the DEFFN statement is a **variable**. It means that this function has been defined (by you) as Function A. It might be helpful to think of this as the name of the particular formula you are using. The **name** you give to the formula must always be included where the **A** is shown. Most programmers save memory by using ONE-LETTER VARIABLES, but the function name can be any legal numeric variable name, up to **FIVE CHARACTERS LONG**. The name must start with a letter but can also include numbers. Here are some function name examples:

Legal DEFFN Names:		Illegal DEFFN Names:
DEFFNP10	DEFFNWAR2	DEFFN1 (number)
DEFFNAB	DEFFNGAME	DEFFN82M(starts w number)
DEFFNABCD	DEFFNR2D2	DEFFNABCDEF (too long)

The DEFFNA is the name of the DEFINE FUNCTION statement. The variable in the parentheses (X) is the NUMBER we are going to manipulate or include in our calculation. DEFFNA(X) simply means we are defining a function called A which is going to manipulate a number called X. Using an X is just a formality. Even if we use X here, later on you can put ANY NUMERIC VARIABLE in the formula by referencing FNA(B2), FNA(B+C+D), FNA (XY) or any other variables in your program. (see ADDING MACHINE with ROUNDING, BELOW). You aren't limited to using the X variable. In other words if you use X or Y or B2 or whatever in your DEFFN formula setup, you can plug other variables in the formula by using FNA (your other variable here), because the A in the FNA activates the formula.

The next part of our DEFFN example is the **equal sign.** This means we are setting FUNCTION A equal to a formula we are going to type on the RIGHT SIDE OF THE EQUAL SIGN. This formula will control what the function actually does.

The right side of the equal sign consists of two parts . . . the **formula or calculation** we want to perform, and the **place** where a number can be inserted. The number is represented by the variable X in our first example and sample format, but it can be any **legal numeric variable name** (a letter, 2 letters, a letter and a number or 2 letters and a number). Where we put our number in the formula (where we put our X) is important because this number will be the KEY to our calculation. In the first example—our 10 percent solution—we made the formula very simple. We multiplied X times 10 percent. Now, every time the VIC encounters an X we can tell it to multiply it times 10 percent simply by including FNA(X) in our program.

Celsius to Farenheit Using DEF FN

Here's an example of a DEF FN formula which converts Farenheit into Celsius . . . we'll call the function C (for Celsius) and call the Farenheit number in the formula F. The function itself looks like this:

```
DEFFNC(F)=5*(F-32)/9
```

Celsius temperature equals 5 times the (Farenheit temperature minus 32 degrees) divided by 9. We've set up a function statement which does just that. Now all we have to add to our program is a means for us to ENTER the Farenheit number. This is very similar to our 10-percent program above:

```
10 DEFFNC(F)=5*(F-32)/9
20 PRINT"ENTER FARENHEIT":INPUTF
30 PRINT"FARENHEIT"F"IS"FNC(F)
      "DEGREES CELSIUS"
```

The structure is simple. First we set up our formula, a straightforward temperature conversion. We call our statement C. The number we want to convert is F for Farenheit.

Next, we PRINT a message telling the user to type an input.

The INPUT F tells the VIC to display a question mark and wait for the use to type (input) a number. As soon as the user types a number, it is automatically given the variable code name "F" (see previous section).

Finally, we PRINT a message which includes the calculation answer. We do this by PRINTing a verbal message INSIDE the quotation marks, then we PRINT the value of F (the number the user typed in from the keyboard) OUTSIDE the quotation marks. Next, we go INSIDE quotes again to type the word "IS", then back OUTSIDE to print the VALUE of FNC(F)—which means the value of F after plugging it into our temperature conversion function statement C in line 10. That value is PRINTed and we're back INSIDE quotes again for the remainder of the message (DEGREES CELSIUS).

You can dress up this program considerably by adding color (after any quotation mark, hold down the CTRL key and hit one of the keys with a color on the front), or REVERSE (inside quotes just like color except press CTRL and RVS ON, then CTRL and RVS OFF to get back to normal).

Here's a SOUND EFFECT you can add to this program. Just type in this line and hit RETURN, then type RUN:

```
25 POKE 36878,15:FORS=200TO125STEP-1:
     POKE36876,S:NEXT
```

This line sets the volume control at 15 (highest), then creates a loop with a musical note range from 200 to 125 and steps down one at a time when the program hits POKE36876,S:NEXT. You should recall from the VIC owner's guide that 36876 is one of the VIC's speaker numbers.

A DEF FN Rounding Program

Here's a program that demonstrates the DEF FN statement, and also provides a "rounding" program. To use it, you have to use a variable (in this case X) to stand for your number. Then, whenever your number, or any number designated X, is generated, it can be **rounded** to the nearest 10th, 100th, 1000th or whatever simply by changing the value of P in line 20. Change P to 10 to round in 10ths for example. Here is the sample program:

```
10 DEFFNA(X)=INT(X*P+.5)/P
20 P=100
30 PRINT"ENTER A NUMBER":INPUTX
40 PRINT"YOUR NUMBER ROUNDED TO THE
     NEAREST"P"TH IS"FNA(X)
50 GOTO10
```

A Simple Adding Machine

Here's a simple program using the INPUT statement to create a simple adding machine:

```
 5REM ADDING MACHINE
10INPUTA:PRINTC+A
20INPUTB:PRINTC+A+B
```

```
30C=A+B+C
40A=0:B=0
50GOTO10
```

Here's how this program works . . . first we INPUT a number which we call A. Then we PRINT the value of C and A, which on the first round does nothing (because C is not yet assigned a value) and only the A number is PRINTed.

In line 20 we INPUT the value of the second number, which we call B. Then we PRINT the sum of A+B (our adding machine) but again no value is assigned to C yet so the C is ignored and we get the sum of the first two numbers.

In line 30 we create a new variable C and define C as the sum of A and B. In other words, C now becomes the value of the first two numbers added together.

In line 40 we reset A to zero and B to zero. This is called "clearing our variables" and means here that we put A and B back to zero so we could type in NEW VALUES for both of these numbers. The sum of the two numbers we already typed in is preserved in our new variable C.

Now in line 50 we go back to line 10 and repeat the program, except NOW the C variable has a value (the sum of the first two numbers we typed in). So when we type in a new number called A (INPUT A), the VIC PRINTs the sum of A (the new number) and C (which we define as the sum of our first two numbers). The result is the added sum of the first three numbers entered.

In line 20 we enter the next number (INPUT B) which we add to our new number (A) and our old sum (C). The total is PRINTed.

Line 30 redefines C. This time, C becomes the total of the old sum (C), our new number A, and our new number B. Now we can set A and B back to zero and start over again.

More specifically . . . if we were adding 10 every time, we would start by adding 10 (A) to 10 (B) which then becomes 20 (A+B). C is then defined as 20 and A and B are set back to zero. Then the program repeats to add a new number 10 (new A) to 20 (C) to get 30 . . . then adds 10 (new B) to 30 (C) to get 40, then redefine C as 40, or 10 (new A) + 10 (new B) + 20 (old C).

Adding Machine With Rounding Function

If we want to add decimal point numbers and ROUND them off, we can do so by incorporating the ROUNDING function described previously. Here's how we would add the rounding function to our ADDING MACHINE program to round off all numbers to the nearest 100th (2 decimal places rounded up or down):

```
 5REM ADDING MACHINE WITH ROUNDING
10DEFFNR(A)=INT(A*100+.5)/100
20INPUTA:PRINTFNR(C+A)
30INPUTB:PRINTFNR(C+A+B)
40C=A+B+C
50A=0:B=0
60GOTO20
```

This rounding program takes our A, B and C variables and plugs them into the DEFFN rounding formula we described earlier. Now ANY value put in the parentheses after DEFFNR( ) will automatically be rounded and plugged into the formula, because it's the DEFFNR that determines what is done to the information in the parentheses, and after the FIRST definition of the function (line 10) any time we want to plug a variable into the formula, we can do so by typing DEFFNR followed by the variable we want to manipulate in parentheses.

In the program above, notice that we only had to type an A in parentheses in line 10. Later, when we type in line 20: FNR(C+A) the program automatically inserts the C+A variables where the A was in line 10. The C+A variables could have been any other numeric variables we might use in our program—M, R, XX, S2, etc.

Scientific Notation, Binary Decimal & Other Peculiarities

Because the VIC calculates using the binary number system, and translates it into our normal decimal numbering system, a few peculiarities may arise . . . for example, if you type: PRINT.34−.30 you will get an answer that looks like this: .0400000001

Clearly, the extra decimal places and the last 1 do not belong in our answer, which should be .04. The difference is so minimal that it doesn't affect most calculations. The best way to avoid this discrepancy is to use the rounding routines discussed earlier.

Another mathematical quirk is if you type: PRINT .5555555556 The VIC will PRINT .555555555 and lose the 6. The VIC rounds DOWN automatically at six or less digits, and rounds UP if there are seven or more digits. This results from the way computers store floating point numbers.

Another idiosyncrasy of the VIC is its use of **scientific notation** when an answer or calculation exceeds a certain limit. This special notation allows the VIC to display large numbers using fewer digits, and is used by many computers. Scientific notation takes the form:

numberE+ee

Here are some examples:

Standard	Scientific Notation
20	= 2E+1
10500	= 10.5E+3
6600	= 66E+2
.66	= 66E−2
.0000000001	= 1E−10

Summary

If you are planning to use your VIC for calculations, these notes to supplement your user's manual should help. The *VIC 20 Programmer's Reference Guide* provides additional information, and most BASIC programming manuals for the PET/CBM or VIC will give you more insights.

Using Printers with the Waterloo microSystems

1. INTRODUCTION

There are several methods for using printers with the Waterloo microSystems on a Commodore SuperPET or Northern Digital microWAT. Some printers are attached via the RS-232 serial port, others via the IEEE-488 bus. Problems arise because of the differences between character representations supported by the SuperPET. Commodore software and Commodore printers, such as the CBM 4022, use a unique character set termed 'PET ASCII'. Waterloo software and most computing equipment manufacturers use standard ASCII for compatibility with other computers with which the SuperPET is capable of communicating.

Several ASCII printers are available with IEEE-488 bus adapters. Confusion arises when these are used as if they were Commodore printers. Output is directed to Commodore printers using file (device) name 'printer'. Translation from standard ASCII to PET ASCII is performed by the microSystems software when this file name is used. However, this translation will cause surprises if the printer is an ASCII device. If an ASCII printer is attached to the IEEE-488 bus, the filename must be 'IEEEX' where 'X' is replaced by the primary address (e.g. 'Ieee4'). When the output is sent to this filename, it will not be translated.

When printers are attached to the RS-232 port, the filename must be 'serial'. The RS-232 port can be used to attach serial printers or terminals such as the AJ832 or Diablo Multiwriter.

2. EXAMPLES USING PRINTERS

This section describes using printers with the various micro languages. Both Commodore printers and standard ASCII devices are explained.

2.1 microAPL

The Commodore 4022 printer cannot print APL characters, but can be used to print alphanumerics such as a data listing or program output.

The first example uses the Commodore printer. The program opens the printer for write, prints the 'message', and then closes the printer.

```
'PRINTER' ]CREATE 1
(]XR 'message',]TC[7]) ]PUT 1
      .
      .
      .
]UNTIE 1
```

In the second example, the printer is attached via the RS-232 serial line. The device is opened, the function 'F' is listed, and then the device is closed. APL characters can be printed with a device such as the Diablo Multiwriter.

```
'SERIAL' ]CREATE 1
(]XR ,(]CR 'F'),]TC[71]) ]PUT 1
      .
      .
      .
]UNTIE 1
```

Should an ASCII device require connection to the IEEE=488 bus, the open statement would be:

```
'IEEE4' ]CREATE 1
```

2.2 microBASIC

```
010 !Example printing on a
    Commodore printer
020
030 open #2,'printer',output
040
050 loop
060     print 'Input name and age'
070     input name$,age
080     if name$='quit' then quit
090     print #2,name$,age
100 endloop
110
120 close #2
130 stop
```

In statement 030, the file 'printer' is opened for 'output'. The #2 in the open statement is a unit number assigned to the file for this program only. All references to the file 'printer' in this program are done using this number rather than the file name.

The program requests the user to type a name and age at the terminal. This information is then printed on the Commodore printer attached via the IEEE-488 bus. When the name 'quit' is entered the program halts.

If the output device were an ASCII printer attached via the RS-232 serial line, the only change would be to 'open' statement:

```
030 open #2,'serial',output
```

Similarly, if the device were an ASCII printer attached via the IEEE-488 bus, statement 030 could be coded as:

```
030 open #2,'ieee4',output
```

2.3 microCOBOL

The following is a Waterloo microCOBOL program that writes to a Commodore printer.

```
identification division.

program-id. XXXX.

environment division.

configuration section.

source-computer. CBM-SUPERPET.
object-computer. CBM-SUPERPET.

Input-output section.

file control.
   select printer-device
   assign to 'printer'.

data division.

file section.

fd printer device
   label records are standard
01 display-record.
   02 filler   picx(80).

working-storage section.
```

```
01 str pic x(80) value is
      'A sample program that writes to
   the printer'.

procedure division.

   open output printer-device.
   write display-record from str.
   close printer-device.
   stop run.
```

The file-control section assigns the file name 'printer' to the file variable 'printer-device'. The string to be printed is assigned to the variable 'str' in the working-storage section. The 'printer-device' is then opened for output, the 'str' is printed, and the device is then closed.

If a serial line device were used, the only change required would be to the file-control section. This would become:

```
file-control.
   select printer-device
   assign to 'serial'.
```

If the device were an ASCII printer attached via the IEEE-488 bus, the value 'serial' would be replaced by 'ieee4'.

2.4 microFORTRAN

The same example that was used for microBASIC is coded in microFORTRAN.

```
* Using a printer with microFORTRAN

integer age
character name

open (unit=2,file="printer")

loop
   print,"Input name and age"
   read,name,age
   quitif name = "quit"
   write(unit=2) name,age
endloop

close (unit=2)
end
```

Again, to use a different printer the only change required is the 'open' statement. To use an ASCII printer attached via the RS-232 port, the statement becomes:

```
   open (unit=2,file="serial")
```

To use the IEEE port with a printer other than Commodore, the statement becomes:

```
   open (unit=2,file="ieee4")
```

2.5 microPascal

In microPascal, there are two ways to handle accessing a printer. This example simply prints the string 'A sample program to write to a printer'.

```
program example1(output,device);

var
   device:text;

begin
   rewrite(device,'printer');
   write1 n(device,'A sample program
   to write to a printer');end
```

The first statement is a program heading which gives a name (example1) to the program followed by a parameter list. Here the file 'device' is listed as an external file.

Within the main block, the file 'device' is declared to be of type 'text'. Variables of type 'text' are referred to as textfiles. Textfiles have the property that they may be divided into lines.

The 'rewrite' statement takes a file variable as a parameter and initializes the file for writing. Here the file variable 'device' has the value 'printer'. If the output device were attached to the RS-232 port, the 'rewrite' statement would become:

```
   rewrite(device,'serial')
```

Similarly, if the device were IEEE compatible but not a Commodore printer, the 'rewrite' statement would appear as:

```
rewrite(device,'ieee4')
```

The 'writein' statement indicates the file variable 'device' which may have a value of 'printer', 'serial', or 'ieee4' depending upon what was assigned in the 'rewrite' statement. The string to be printed follows the file variable. The line that is printed will be ended and should another write or writein statement follow, that output string would appear on the following line.

The 'end' statement completes the block.

A second method of accessing the printer is to use the standard output file as follows:

```
program example2(output);

begin
   rewrite(output,'printer');
   writein('A sample program to write
   to a printer');

end.
```

The standard file 'output' is an external file indicated as a parameter in the program heading. The following declaration is assumed automatically if the file is mentioned in the program heading.

```
var
   output : text
```

The file 'output' is automatically initialized before program execution is started; i.e. rewrite(output). However, in this example, the 'rewrite' statement is included to assign the value 'printer' to the file variable 'output'

The procedure 'writein' assumes the standard file 'output' if the optional parameter specifying the file is omitted. Hence the 'writein' statement contains only the string to be printed.

Again, the value 'printer' in the 'rewrite' statement could be replaced by 'serial' or 'ieee4' to use printers other than a Commodore printer.

example is function key 3. When hit, then letters R U N are printed on the screen and a 'CARRIAGE RETURN' is performed. Thus one keystroke will run your program. F6 prints 'RIGHT$(' on the screen.

★ PROG and EDIT are commands which change the values obtained through the function keys. Commands while in PROG mode obtainable from the function keys include GOSUB, RETURN, MID$, CHR$ etc. EDIT mode gives AUTO, DELETE, TRACE, MERGE etc. As I say, these may be changed by the user, but the command PROG or EDIT will cause the user's functions to be erased. These keys may also be used in INPUT or GET statements: the result is exactly as though the word had been typed from the keyboard.

★ PROGRAMMERS' AID also provides control functions from the keyboard. CTRL L will clear the characters on the screen from the cursor to the end of the screen line. As you know, the screen line may be 22, 44, 66 or 88 characters long. CTRL N will clear the screen from the cursor to the end of the screen. CTRL U will clear all the characters on the line on which the cursor is sitting and return the character to the first position on that line. CTRL E will escape from quote mode. That is, if pressed while inside inverted commas, cursor control characters are obeyed instead of the RVS code being printed. It also escapes from the quote mode created by the INST key.

CTRL Q and CTRL A are whizz-bang little tools! Think of all the times you LIST your program only to have the line you want scroll off the top of the screen. Now, you can CTRL Q, and the cursor will move to the top of the screen. If you keep your fingers on CTRL Q, the listing magically scrolls DOWN to look at previous lines! CTRL A does the same thing but allows scrolling through the later lines of the program.

I think the manual is written in 'Jinglish' – translated Japanese – but is otherwise understandable. Quite a few typographical errors, but the general meaning is there. The cartridge itself can be extremely invaluable in certain circumstances, and while, in my opinion, there are a few deficiencies, there are also a few quite nice features. I won't start complaining about the fact that the advance publicity for the VIC stated that machine code monitor functions would be included in the Programmers' Aid Cartridge, but suffice to say I found their absence disappointing.

Both experienced programmers and beginners would eventually recover the purchase price in not having to pay hefty psychiatrist bills. So just think of it as an investment in sanity.

SUPER DRAW.

I can do some wonderful things with my super expander, in or out of the VIC! But the one thing I cannot do with it is to draw pretty pictures on my television screen. I also play with my joystick and have lots of fun. But what else could I do with it?

What about designing a program to incorporate both. Well, many days later there it was, and believe it or not, all in just the super expander and no extra memory.

The structure of the program is quite simple. Once the program is RUN it takes over. From here on, there are no prompts, so the user must know what the computer is expecting. For instance if 'C' for change colours is selected, then the graphics cursor would disappear and the computer will wait for four colour values to be input, each followed by return.

There are two main sections of the program, a section which flashes a cursor on the screen until the joystick button or a key is pressed and secondly a routine which plots with a flashing cursor until the button is pressed or a key is pressed. If the button is pressed no point will be plotted. If a key is pressed it goes back to cursor mode via the menu in PLOT mode.

All the other options such as the draw routine use either the cursor or plot routines.

Bugs So Far.

The program was designed with the use of multi-colour mode in mind. The program will operate as it is in multi-colour mode just by changing the one and only GRAPHIC statement in line 120. The reason this was not done is because the 'load' and 'save' options do not save the extra information regarding colour, as would be required for multi-colour mode.

One possibility is to incorporate a sequential file which stores this information into the program. Maybe someone could send in a copy that does this – or something similar.

Possible Additions.

Something that would be of great use would be the facility to speed up the cursor movement. The simplest way to do this would be to add a variable which could be added or subtracted to the present cursor position in lines 1030-1045 and 6030-6045. This then could be added as another function, all that the module would need to do would be to add or subtract '6' from this variable.

Useful Routines.

Three 'neat' little routines are used in this program which I think deserve extra credit.

Lines 3050 to 3110 contain a routine which saves a portion of memory onto any device. In this case, it is used to save the graphic's screen as a program file. Oops, there goes another bug, if you have a cassette then you will need to change lines 3060 and 4060.

Routine number two in lines 1540 and 1550 does something that the Super Expander should do. That is find out whether a dot on the screen is on or off. Given 'CX' and 'CY' the routine returns 'D' equal to 1 or 0 depending if the dot is on or off. The super expander does have the RDOT function which is very similar, but did not meet the situation.

The way in which all the inputs are done also deserves a mention. Notice that they are all done by the use of the INPUT statement; so be careful with the cursor keys.

The Super Expander does not use all the screen as you may know; it only uses the first 10 full screen lines. So, by positioning the cursor at the 13th line (just to be sure) we can execute a normal INPUT statement. The variable PO$ does the positioning while CL$ will clear the same line ready for next time.

```
READY.

10 KEY1,"P":REM PLOT
20 KEY2,"D":REM DRAW
30 KEY3,"F":REM FILL
40 KEY4,"R":REMREGION
50 KEY5,"C":REM COLOR
60 KEY6,"N":REM PEN
70 KEY7,"S":REM SAVE
80 KEY8,"L":REM LOAD
90 REM "■"=CLR SCREEN
100 REM
110 REM SET-UP
120 IFFL=0THEN:GRAPHIC2:CLR:FL=1
130 R=0:X1=0:Y1=0:CX=510:CY=510
140 CL$="■■■■■■■■■■■■■■■":PO$=CL$:CL$=CL$+"                "+"■"
150 SC=1:BO=1:CH=4:AU=1:GOSUB5500:PN=1
200 REM
210 REM DEFAULT MODE (CURSOR,PLOT WITH BUTTON DOWN)
220 GOSUB1000:REM CUSOR ROUTINE
230 IFA$<>""THEN300:REM GO TO MENU
240 CL=CH
250 GOSUB2000:REM PLOT POINT
260 GOTO220
300 REM
310 REM MAIN MENU
320 IFA$="L"THENGOSUB4000:REM LOAD ROUTINE
330 IFA$="S"THENGOSUB3000:REM SAVE ROUTINE
340 IFA$="C"THENGOSUB5000:REM CHANGE SCREEN COLOURS
350 IFA$="P"THENGOSUB6000:REM PLOT MODE
360 IFA$="N"THENGOSUB7000:REM CHANGE PEN TYPE
370 IFA$="F"THENGOSUB8000:REM FILL IN SHAPE
380 IFA$="R"THENGOSUB9000:REM DO REGION CHANGE
390 IFA$="D"THENGOSUB9500:REM GO INTO DRAW MODE
400 IFA$="■"THEN:SCNCLR:REM CLEAR SCREEN
410 GOTO 200
1000 REM
1005 REM CURSOR
1010 GOSUB 1540
1020 POINT1,CX,CY:GETA$:R=RJOY(0):IF(R=0ANDA$="")THEN:POINT0,CX,CY:GOTO1020
1030 IFRAND1THENY1=-6:GOSUB1500
1035 IFRAND2THENY1=+6:GOSUB1500
1040 IFRAND4THENX1=-6:GOSUB1500
1045 IFRAND8THENX1=+6:GOSUB1500
1050 IF((RAND128)=128)ORA$<>"")THENGOSUB1530:RETURN
1060 GOTO1020
1500 IFCX=6THENX1=+6
1505 IFCX=1020THENX1=-6
1510 IFCY=6THENY1=+6
1515 IFCY=1020THENY1=-6
1530 REGION CL:POINTD,CX,CY:CX=CX+X1:CY=CY+Y1:Y1=0:X1=0
1540 X=CX/6.4:Y=CY/6.4:X%=X/8:Y%=Y/16:P=PEEK(X%+Y%*20+7680)
1550 I=4096+P*16+(YAND15):D=0:C=(2↑(7-(XAND7))):IF(PEEK(I)ANDC)=CTHEND=1
1560 IFD=0THENRETURN
1570 CL=RDOT(CX,CY):RETURN
2000 REM
2010 REM PLOT POINT
2020 REGIONCH:POINTPN,CX,CY:RETURN
3000 REM
3010 REM SAVE PAGE
3020 PRINTPO$;
3030 INPUTA$
3040 PRINTCL$
3050 POKE183,LEN(A$):FORI=1TOLEN(A$):POKE819+I,ASC(MID$(A$,I,1)):NEXT
3060 POKE186,8:POKE185,1:REM DEVICE ADDRESS
```

```
3070 POKE187,52:POKE188,3:REM WHERE FILENAME
3080 POKE172,0:POKE173,16:REM START ADDRESS
3090 POKE174,128:POKE175,28:REM END ADDRESS
3100 POKE193,0:POKE194,16:REM LOAD ADDRESS
3110 SYS 63109
3120 RETURN
4000 REM
4010 REM LOAD PAGE
4020 PRINTPO$;
4030 INPUTA$
4040 PRINTCL$
4050 S=37888:F=S+1023:FORI=STOF:POKEI,CH:NEXT:FL=1
4060 LOADA$,8,1
4070 REM RUNS FROM START NOW
5000 REM
5010 REM INPUT COLOR OPTIONS
5020 PRINTPO$;
5030 INPUTSC:PRINTCL$PO$;:INPUTBO:PRINTCL$PO$;
5040 INPUTCH:PRINTCL$PO$;:INPUTAU:PRINTCL$;
5500 COLORSC,BO,CH,AU:RETURN
6000 REM
6010 REM SET UP PLOT - WHILE BUTTON IS NOT DOWN
6015 GOSUB 6560
6020 POINT1,CX,CY:GETA$:R=RJOY(0):IF(R=0ANDA$="")THEN:POINT0,CX,CY:GOTO6020
6030 IFRAND1THENY1=-6:GOSUB6500
6035 IFRAND2THENY1=+6:GOSUB6500
6040 IFRAND4THENX1=-6:GOSUB6500
6045 IFRAND8THENX1=+6:GOSUB6500
6050 IFA$<>""THENGOSUB6530:RETURN
6060 GOTO6020
6500 IFCX=6THENX1=+6
6505 IFCX=1020THENX1=-6
6510 IFCY=6THENY1=+6
6515 IFCY=1020THENY1=-6
6530 IF(RAND128)<>128THEN:POINTPN,CX,CY:GOTO6550
6540 POINTD,CX,CY
6550 CX=CX+X1:CY=CY+Y1:Y1=0:X1=0
6560 X=CX/6.4:Y=CY/6.4:X%=X/8:Y%=Y/16:P=PEEK(X%+Y%*20+7680)
6570 I=4096+P*16+(YAND15):D=0:C=(2↑(7-(XAND7))):IF(PEEK(I)ANDC)=CTHEND=1:RETURN
6580 RETURN
7000 REM
7010 REM INPUT PEN TYPE
7020 PRINTPO$;
7030 INPUT PN
7040 PRINTCL$
7050 RETURN
8000 REM
8010 REM FILL IN SHAPE
8020 GOSUB 1000 :REM CURSOR ROUTINE
8030 PAINTPN,CX,CY:RETURN
9000 REM
9010 REM CHANGE REGION COLOUR
9020 PRINTPO$;
9030 INPUTCH
9040 PRINTCL$
9050 RETURN
9500 REM
9510 REM DRAW ROUTINE
9520 AX=0:AY=0:BX=0:BY=0
9530 GOSUB 1000
9540 IF A$<>""THENRETURN
9550 AX=CX:AY=CY:POINTPN,AX,AY
9555 GOSUB1000:IFA$<>""THENRETURN
9560 BX=CX:BY=CY:GOSUB9800:AX=BX:AY=BY:GOTO9555
9800 DRAWPN,AX,AYTOBX,BY:RETURN
READY.
```

EDUCATIONAL COMPUTING

As an educator, I have my reservations about microcomputers as teaching machines. For a start, most of the 'educational' programs I have seen have been written by people who are not trained teachers. You don't need to be trained to be a good teacher, but it helps. For a start, when you have been at it a while, you realise that there is more to teaching than dull repetition.

And yet this is the basis of so many of the programs that appear in computer magazines. Maybe Fond Parent, having blown the housekeeping (and then a bit more!) on a micro, feels the need to show that really, after all, when all is said and done (etc.) it is All for the Good of the Children. And so we are subjected to yet another set of computerised flash cards, or another 'Hangman', or another maths drill package.

I suspect that, next century, it will be recognised that the most rediscovered wheel of this century is the maths drill package. The approach is obvious: two random numbers, in a defined range. First, Junior has to add, subtract, multiply or divide the numbers. Then the program checks the answer.

Then we add a little bit of sound, colour, graphics display or whatever to reward Junior when he/she gets it right, and that's it!

All of this may help us to get our personal pedagogical rocks off, but what does it do to Junior? Maybe this is why Junior's eyes shine at the mention of Ned Ludd?

This problem was recognised by Marshall McLuhan, who warned us against 'rearview mirror driving': interpreting a new medium in terms of old experience. To put it another way, the important things to consider about any medium are:

1. The strengths of the medium;
2. The weaknesses of the medium; and
3. The ways in which these weaknesses and strengths may be exploited.

The rearview mirror approach looks only at the weaknesses of the medium. Teachers become impatient when faced with repetitive drilling of recalcitrant Juniors, so the new computer medium is adopted to do what teachers can't or won't.

But maybe drilling is wrong, or ineffective. Maybe it is a good thing that teachers avoid too much drilling. Just because we had to endure something, why should our kids?

If we look ahead, rather than in the rearview mirror, we will ask: what do microcomputers do well? True, they are good at repetitive tasks, but they are also good at displaying results, sorting through information, manipulating strings, and so on.

Another tack would be to think more about other sorts of education. Maybe we can teach computing! And so we come to my first offering, a small program which teaches something about the way in which a computer works (Listing 1).

But first, a small digression about what I was actually doing when I wrote the program, because the program was NOT written with the education of others in mind.

There are a number of packages running on mainframes which analyse survey data, producing cross-tabulations. I know of no such programs written for microcomputers, and I wondered why. Try DIMensioning A(35,400), and you will see why: the overhead costs of two-dimensional arrays are just too great. So I started thinking about Boolean algebra, and the possibility of storing several values in one byte.

The trouble lay in the weakness and staleness of my Boolean algebra. I needed a program that would explore and explicate the recesses of that which Boole wrought.

I found this program useful in working out my own thinking, although the cross-tabulation program is still a Future Thing. But on the side, it seemed to me that the program has a useful educational function, letting the user see just what ANDing does. The program can also be modified by changing each AND to OR. Youngsters are amused to find the computer making apparently bizarre statements about what '5 AND 43' equals, but the display of the binary equivalents of the numbers makes understanding easier to acquire.

```
LISTING 1

READY.

 10 DIM X$(15): PRINT "□"
 20 INPUT "□FIRST NUMBER ";A: X = A: GOSUB 60
 30 INPUT "SECOND NUMBER ";B: X = B: GOSUB 60
 40 X = A AND B:PRINT "□";A;" AND";B;" = ": GOSUB60
 50 PRINT "□";A;" AND";B;" =";  (A AND B): GO TO 20
 60 PRINT TAB(8)"BINARY VALUE ";:FOR I = 1 TO 15
 70 IF (X - 2↑(15-I) => 0)THEN X = X - 2↑(15-I): X$(I) = "1": GO TO 90
 80 X$(I) = "0"
 90 PRINT TAB(22) X$(I);: NEXT: PRINT: RETURN
READY.
```

The program was not written as an 'educational' program. It was written to explore an aspect of BASIC which I did not fully understand, and then developed somewhat to make the rather neater parcel I present here. I would suggest that a great many useful 'computer educational' demonstrations arise in this way. I do concede that some programs will occur which are written with teaching in mind, but I just don't believe that you can sit down at the keyboard and 'write something educational'. You need to have some idea of what effect you want.

My second offering was deliberately conceived and written as a 'game' which would help to familiarise my six-year-old daughter with the use of the numeric as a directional control in 'CURSOR' games.

One of the 'games' that my kids both enjoy is 'Printsit' on 'CURSOR 24'. This makes a pattern on the screen (and later prints it out). The numeric pad is used to move the cursor around the screen. Having once discovered the uses of PEEK(151) to make a sort of auto-repeat, I quickly added this to 'Printsit', and away we went.

Then I found a short machine language sequence in another CURSOR program that reverses the screen. And so 'Pattern Shaker' was born: I deleted the printer, it was costing me too much in paper. Then I developed a 'bare bones' auto-repeat cursor control from the numeric pad, and I made some of the cursor control keys do odd things, mainly as suggested by my eight-year-old son. I do not propose to detail these here, as exploration will be its own reward. One thing I will note: for some reason the machine language routine seems to get degraded from time to time, and I don't know why.

So be on the safe side, I RESTORE and POKE the whole lot back in each time I want to use the sequence. Lazy and clumsy, I know, but better than getting hung up.

So there we have it: two 'educational' programs, neither of which is a maths drill package. The first should teach a better understanding of computer logic, and the second should help to foster a more confident approach to the keyboard.

There is one other way in which these programs might be 'educational': maybe when they look at the programs, kids will notice some handy techniques. Now next issue, if the editor has room, I would like to discuss my maths drill package that incorporates 'Hangman' on flashcards.....

```
LISTING 2
READY.
 0 GOTO20
 1 IF(C>1ANDR<25)THENC=C-1:R=R+1:REM1
 2 RETURN
 3 IFR<25THENR=R+1:REM2
 4 RETURN
 5 IF(C<40ANDR<25)THENC=C+1:R=R+1:REM3
 6 RETURN
 7 IFC>1THENC=C-1:REM4
 8 RETURN
 9 IFC<40THENC=C+1:REM6
 10 RETURN
 11 IF(C>1ANDR>1)THENC=C-1:R=R-1:REM7
 12 RETURN
 13 IFR>1THENR=R-1:REM8
 14 RETURN
 15 IF(C<40ANDR>1)THENC=C+1:R=R-1:REM9
 16 RETURN
 20 PRINT"□":A=32727:C=1:R=1:W=40:D=25:P=A+R*W+C:V=102:POKEP,V:POKE59468,12
 30 GETA$:IFA$=""THENFORI=1TO160:NEXT:POKEP,160:FORI=1TO160:NEXT:POKEP,V:GOTO30
 40 V=ASC(A$):IFV>48THENIFV<58THENV=V-48:GOTO500
 50 IFV=29THEN700
 60 IFV=17THENGOSUB600:SYS(826):RESTORE:GOTO30
 70 IFV=145THENGOSUB600:GOTO610
 80 IFV=147THENPRINTCHR$(147):GOTO30
 90 IFV=19THENR=1:C=1:P=A+R*W+C:GOTO30
 100 IFV=20THENFORI=32768TO33767:POKEI,V:NEXT:P=32768:R=1:C=1:GOTO30
 110 V=VOR64
 120 V=V:POKEP,V:GOTO30
 500 ONVGOSUB1,3,5,7,2,9,11,13,15
 510 P=A+R*W+C:POKEP,V:IFPEEK(151)=255THEN30
 520 GOTO500
 600 FORJ=826TO858:READB:POKEJ,B:NEXT:RETURN
 601 DATA169,128,141,72,3,141,77,3,160,4,162,0,189,0,128,73,128
 602 DATA157,0,128,232,208,245,238,72,3,238,77,3,136,208,234,96
 610 FORI=1TO100:SYS826:FORJ=1TO100:NEXT:NEXT
 615 REM: BE WARY OF USING THIS ROUTINE WITH EPILEPSY SUFFERERS.
 620 RESTORE:GOTO30
 700 FORI=32768TO33767:IFPEEK(I)=96ORPEEK(I)=32THEN720
 710 POKEI,V
 720 NEXT:P=1:C=1:P=A+R*W+C:GOTO30
READY.
```

THE COMPUTERISED HISTORY OF AUSTRALIA

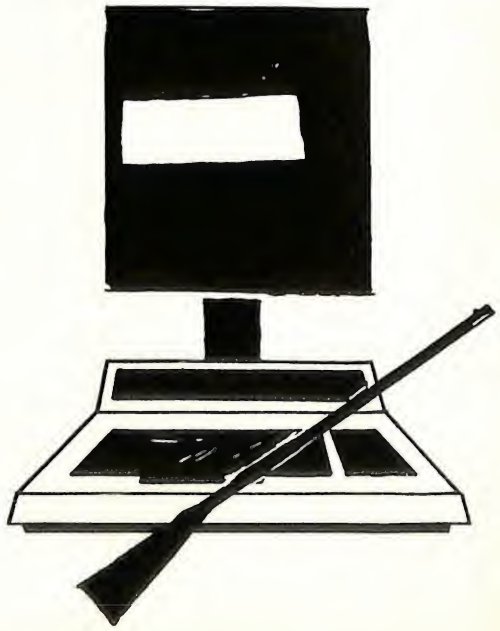
PROGRAM FOR 4000 & 8000 SERIES

by Peter Macinnie

```
READY.

 1 GOTO10
 2 FORI=A TO B STEP C:FOR J=0 TO E:POKE(I+J),160:NEXTJ:NEXTI:RETURN
 3 FORI=A TO B STEP C:POKEI,32:NEXT:RETURN
 4 FORI=A TO B STEP C:POKEI,105:NEXT
 5 FORI=A+1 TO B+1 STEP C:POKEI,233:NEXT:RETURN
 6 FORI=A TO B:POKEI,160:NEXT:RETURN
 7 FORI=ATOBSTEPJ:POKEI,160:NEXT:RETURN
 10 PRINT"■":POKE59468,12:PRINTTAB(15)"OZPRINTER."
 20 PRINT"■A SHORT HISTORY OF AUSTRALIA IN PICTURES"
 30 PRINT"■■PROGRAMMED BY PETER MACINNIS,":PRINT"6 BOYLE STREET, BALGOWLAH,
 40 PRINT"NEW SOUTH WALES, 2093.
 50 PRINT"■■LINES 59000+ ARE FROM 'PRINTSIT',        COPYRIGHT 'CURSOR' MAGAZINE.
 60 PRINT"■PERMISSION TO USE THESE HAS YET TO BE   OBTAINED
 70 PRINT"■IF YOU HAVE NO PRINTER, STOP THE RUN ANDLIST THE REMS AT 60000+
 80 PRINT"■■THEN PRESS ANY KEY TO BEGIN.
 85 GETA$:IFA$=""THEN85
 90 T$="A SHORT HISTORY OF AUSTRALIA"
 99 GOTO150
 100 OPEN4,4:PRINT#4,CHR$(1)T$:PRINT#4,CHR$(13):CLOSE4
 150 S$="                                        "
 190 PRINT"■":FORVV=1TO8:READV:READV$
 199 REM SET UP MAIN BLOCKS OF MAINLAND
 200 A=33018:B=33498:C=40:D=0:E=20:GOSUB2
 210 A=33119:B=33719:E=1:GOSUB2:A=33241:B=33601:GOSUB2
 230 A=33211:B=33451:E=6:GOSUB2:A=33493:B=33573:GOSUB2:A=33552:B=A+40:E=6
 260 GOSUB2:A=32865:B=32945:E=4:GOSUB2:A=32979:B=32990:GOSUB6
 310 A=33250:B=33370:J=40:GOSUB7:A=33173:B=33177:GOSUB6
 330 A=33135:B=A+2:GOSUB6:A=33096:B=A+1:GOSUB6:A=33613:B=A+2:GOSUB6
 360 A=33538:B=A+2:GOSUB6:A=A+11:B=A+1:GOSUB6:A=33633:B=A+4:GOSUB6
 390 A=33674:B=33677:GOSUB6:A=33524:B=33324:J=-40:GOSUB7:A=33563:B=33243
 410 GOSUB7:A=33078:B=32958:GOSUB7:A=32997:B=A-2:J=-1:GOSUB7
 430 A=32915:B=32916:GOSUB6
 440 POKE32915,160:POKE32916,160:POKE 32866,160
 450 A=32955:B=32957:GOSUB6
 460 POKE32876,160:POKE32904,160:POKE32944,160:POKE32943,160
 600 REM INSERT COAST:LINE DETAILS
 610 POKE32903,233:POKE32864,233:POKE32865,247:POKE32826,111:POKE32827,98
 620 POKE32870,160:POKE32910,105:POKE32950,223:POKE32991,252:POKE32992,98
 630 POKE33033,252:POKE32994,233:POKE32875,245:POKE32835,108:POKE32836,98
 640 POKE32877,116:POKE32917,117:POKE33039,223:POKE33079,160:POKE33121,223
 650 POKE33161,160:POKE33162,223:POKE33201,160:POKE33202,160:POKE33203,111
 660 POKE33244,123:POKE33284,252:POKE33285,123:POKE33325,252:POKE33365,234
 670 POKE33405,160:POKE33445,160:POKE33485,105:POKE33564,105:POKE33603,116
 680 POKE33641,160:POKE33642,105:POKE33681,234:POKE33721,126:POKE33720,120
 690 POKE33719,236:POKE33718,160:POKE33717,105:POKE33716,228:POKE33715,239
 695 POKE33638,160:POKE33678,160
 700 POKE33714,120:POKE33673,95:POKE33591,118:POKE33551,118:POKE33590,105
 710 POKE33589,95:POKE33548,95:POKE33507,239:POKE33506,226:POKE33505,120
 720 POKE33504,226:POKE33503,239:POKE33542,119:POKE33541,105:POKE33580,120
```

```
730 POKE33579,226:POKE33578,239:POKE33616,105:POKE33615,160:POKE33614,160
740 POKE33654,120:POKE33613,160:POKE33612,245:POKE33572,245:POKE33532,244
750 POKE33492,160:POKE33491,95:POKE33450,225:POKE33410,244:POKE33370,160
760 POKE33329,95:POKE33289,160:POKE33249,160:POKE33209,98:POKE33210,247
770 POKE33171,121:POKE33172,98:POKE33133,121:POKE33134,98:POKE33095,233
780 POKE33056,244:POKE33016,108:POKE33017,233:POKE32978,233:POKE32939,98
790 POKE33057,160:POKE32940,160:POKE32941,252:POKE32942,248:POKE32943,160
799 REM 800-830AND910-930 SPELL OUT 'OZ'
800 A=33058:B=A+7:C=1:GOSUB3:A=A+40:B=33258:C=40:GOSUB3
810 A=33105:B=33265:GOSUB3:A=33298:B=A+7:C=1:GOSUB3
820 A=33067:B=A+6:GOSUB3:A=33112:B=33307:C=39:GOSUB4:A=B+1:B=A+6:C=1
830 GOSUB3:POKE33074,233:GOTO1020
1020 PRINT"■";S$"■";V$
1030 FORI=32808TO33767:X=PEEK(I):IFX<128THENX=46:GOTO1050
1040 X=V
1050 POKEI,X:NEXT
1060 GOSUB58990:NEXTVV
1100 RESTORE:GOTO190:REM DELETE WHEN PRINTING OR YOU'LL NEVER STOP!!!
2000 DATA48,"AN EMPTY LAND, THEY SAID................."
2010 DATA30,".......ALONG CAME THE CONVICTS.........."
2020 DATA31,"......THEN THEY ALL WENT WEST..........."
2030 DATA122,".........AND WE HAD ONE BIG GAOL........."
2040 DATA43,".........THEN THE BARS BROKE DOWN......."
2050 DATA94,"....A PIE-EATING, HORSE-LOVING NATION..."
2060 DATA42,".........THEN CAME THE BLOW-FLIES........"
2070 DATA83,"...........BUT STILL LOVABLE............"
50000 STOP
58990 GOTO59000:REM: DELETE GOSUB 58990  AT 1060 IF YOU HAVE NO PRINTER
58995 REM PRINT ROUTINE IS FROM CURSOR  MAGAZINE'S "PRINTSIT" PROGRAM.
59000 WD=40:CRT=32768
59005 OPEN4,4:ZC=WD:ZR=0
59006 ZS=23:GOSUB59150:PRINT#4:PRINT#4
59007 ZS=22:GOSUB59150
59010 FORZT=CRTTOCRT+25*WD:Z=PEEK(ZT):ZL=Z
59020 IFZC=WDTHENZC=0:PRINT#4
59030 POKEZT,(ZL+128)AND255
59040 IFZ>127ANDZR=0THENPRINT#4,"■";:ZR=1
59045 IFZ<128ANDZR=1THENPRINT#4,"■";:ZR=0
59050 Z=ZAND127:ZS=0:IFZ>63THENZS=128:Z=Z-64
59060 IFZ<32THENZ=Z+64
59080 PRINT#4,CHR$(Z+ZS);
59090 IFZ=34ANDZS=0THENPRINT#4,CHR$(141);TAB(ZC+1);
59120 ZC=ZC+1:POKEZT,ZL
59130 NEXTZT:PRINT#4:ZS=21:GOSUB59150:PRINT#4
59140 ZS=24:GOSUB59150:PRINT#4:CLOSE4:RETURN
59150 OPEN6,4,6:PRINT#6,CHR$(ZS);:CLOSE6:RETURN
60000 REM: BLOCK LINES 100 AND 1060 IF YOU HAVE NO PRINTER.
60010 REM:DELETE LINES 58990-59150 IF  YOU HAVE NO PRINTER
READY.
```



MAKE THE MOST OF YOUR COMMODORE

HIGH-RESOLUTION GRAPHICS

Now you can give your PET/CBM a High-Resolution Graphics capability with the MTU Graphics Hardware and Software Package. The Hardware is easily installed and the new graphics board provides five EXTRA ROM Sockets and 8k RAM MEMORY EXPANSION which can be used for program or data storage when graphics are not required. A powerful graphics Software Package is included and contains many extra BASIC commands for drawing lines defining shapes etc. The Graphics Hardware does not affect normal operation of the Commodore.

Available on all PET/CBM - BASIC 1, 2, 3, or 4.

RS232 TEST SET

The RS232 TESTSET is a small hand held device that connects inline with the interface cable, the terminal or the modem and monitors the line signals. The TESTSET passes all 25 lines through and so can be left connected without affecting communications. It is completely portable as no batteries are required since power us derived from the interface signals. Each indicator circuit is current limited to meet the requirements of the RS232 Interface Standard. Also the voltage range for activating the bright LED display corresponds with this standard and thereby reduces troubleshooting to a "GO-NOGO" problem instead of trying to measure active signals to determine voltage levels. One 2-pin and one 3-pin jumper are included.

PRINTOUT

Don't forget your PRINTOUT Magazine Subscription - for Commodore PET/CBM Lovers 12 issues p.a. from Jan. '82 incl. postage - $50.00 p.a.

VIC COMPUTING

For all VIC 20 owners, this sister to PRINTOUT is a must. 6 issues p.a. incl. postage from Jan. '82 - $25.00 p.a.

PROGRAMMERS TOOLKIT ROMS

These ROMs plug into spare sockets in your PET/CBMs and give the user additional commands such as TRACE, single STEP, FIND, RE-NUMBER, AUTO line numbering,DUMP variable contents, APPEND, and DELETE multiple lines. Also available are the DISK-O-PRO Tool-kits which gives all the extra DOS 2.0 commands to DOS 1.0 users as well as commands like PRINT USING, SCROLL and disk program MERGE -25 commands in all. The COMMAND-O provides DISK-O-PRO commands for BASIC 4.0 users.

The Programmers Toolkit for BASIC 1.0/2.0/3.0/4.0 users.
DISK-O-PRO Toolkit for BASIC 2.0/3.0 users.
COMMAND-O Toolkit for BASIC 4.0 users.

Tel: (03) 614 1433
614-1551
Telex: AA 30333.

MICROCOMP
MICROCOMPUTER SYSTEMS DESIGNERS

B.S. MICROCOMP PTY. LIMITED,
4th & 3rd Floors,
561 Bourke Street,
Melbourne, 3000.

The Link

THAT JOINS ALL OFFICE FUNCTIONS

Get it all together with the Silicon Office.

STORING AND RETRIEVING INFORMATION – CREATING, EDITING AND PRINTING OF TEXT – MATHEMATICAL CALCULATION – COMMUNICATING INFORMATION LONG DISTANCE.

Silicon Office is the first database management System for Commodore CBM Microcomputers whereby up to six files may be open and accessed simultaneously during a run. It is also the first system which permits intercommunication with fellow machines and user. The Silicon Office turns the CBM 8032 into a secretarial work station capable of emulating any application package the user cares to think of.

Now one program which is continuously and completely resident in the memory of the CBM is capable of performing all functions required to run a small business or office. This can mean anything from Accounting and Stock Control to Word Processing, Statistical analysis, mailing lists and information filing – all at once, if necessary. Combine filing cabinets, ledgers, typewriter and calculator in your office into one efficient unit.

The Silicon Office package comprises of three integrated elements: a sophisticated word processor, a flexible database management system and an option for inter computer communications — all in one memory resident program.

$8490

THIS BUSINESS PACKAGE IS NOW AVAILABLE AND WILL COMPRISE:

- 8023 DOT MATRIX/PSEUDO PRINTER
- COMMODORE CBM 8032 COMPUTER
- 8050 DISC DRIVE UNIT
- 64K ADD-ON MEMORY BOARD (TOTAL 96K RAM)
- THE SILICON OFFICE PROGRAM MASTER DISC
- TWO SECTIONAL A4 MANUALS

This package is available from:

Compute. CBM SYSTEMS

5 PRESIDENT AVE., CARINGBAH ☎ 525 5022